Biosignaalien tiivistäminen

Timo Tossavainen

Tampereen yliopisto
Tietojenkäsittelyopin laitos
Pro gradu -tutkielma
1.10.1999

Tampereen yliopisto
Tietojenkäsittelyopin laitos
Timo Tossavainen: Biosignaalien tiivistäminen
Pro gradu -tutkielma, 93 sivua
Lokakuu 1999

Tiivistelmä

Tutkimuksessa arvioitiin useiden eri tiivistysmenetelmien soveltuvuutta Helsingin yliopistollisen keskussairaalan otoneurologisessa laboratoriossa mitattuihin silmänliikesignaaleihin. Lääketieteelliset mittaukset halutaan säilyttää jatkotutkimusta varten ja biosignaalit vievät muihin mittauksiin verrattuna paljon tilaa. Tallennusresurssien tarvetta voidaan vähentää tiivistämällä tietoa. Tiedon tiivistäminen voidaan tehdä häviöttömästi, jolloin alkuperäinen tieto säilyy sellaisenaan, tai häviöllisesti, jolloin alkuperäisestä tiedosta säilytetään jokin approksimaatio. Alueen aiempi tutkimus on keskittynyt sydänsähkökäyrien tiivistämiseen; tietääkseni tämä tutkimus on ensimmäinen, joka on tehty silmänliikesignaalien tiivistämisestä.

Häviöttömistä menetelmistä tutkittiin universaalikoodausta, sanakirjamenetelmiä LZ77 ja LZW, Huffman-koodausta ja aritmeettista koodausta. Parhaat tulokset saavutettiin aritmeettisella koodauksella ja tutkimuksen aikana kehitetyllä uudella parametroituihin universaalikoodeihin perustuvalla menetelmällä. Häviöllisistä menetelmistä tutkittiin skalaarikvantisointia, differentiaalista pulssikoodimodulaatiota ja muunnoskoodausta; muunnoskoodauksessa tiivistämiseen käytettiin kynnystämistä ja tutkimuksen aikana kehitettyä uutta kvantisointimenetelmää. Häviöllisistä menetelmistä parhaat tulokset saavutettiin muunnoskoodauksella. Uudella kvantisointimenetelmällä saavutettiin selvästi parempia tuloksia kuin kynnystämisellä. Tutkimuksessa saavutetut tulokset sekä häviöllisten että häviöttömien menetelmien tapauksessa ovat samaa luokkaa ja osittain parempia kuin aiemmissa biosignaalien tiivistämistä koskevissa tutkimuksissa ilmoitetut tulokset.

Sisällys

Kiitokset

Kiitokset Jenny ja Antti Wihurin rahastolle, opettajilleni ja työn tarkastajille.

1 Johdanto

Lääketieteessä tehdään potilaista diagnooseja lukuisten erilaisten mittausten avulla. Mittaukset pyritään säilyttämään jatkotutkimuksia varten. Biosignaalit ovat mittauksia, joissa potilaasta mitataan jotain ajan myötä muuttuvaa suuretta. Tunnetuimpia niistä ovat sydänsähkökäyrä, EKG, ja aivosähkökäyrä, EEG. Tyypillisesti biosignaalit sisältävät muihin mittauksiin verrattuna enemmän informaatiota ja vievät tästä syystä enemmän tiedonsiirto- ja tallennuskapasiteettia. Tiivistämisen avulla voidaan informaatio esittää pienemmässä tilassa ja pienentää tiedonsiirto- ja tallennusresurssien tarvetta.

Tässä tutkimuksessa tutkittiin sekä häviöllisten että häviöttömien tiivistämisen menetelmien soveltuvuutta Helsingin yliopistollisen keskussairaalan otoneurologisessa laboratoriossa mitattuihin silmänliikesignaaleihin. Silmänliikesignaalien avulla tutkitaan mm. sisäkorvaan liittyviä tasapainosairauksia; diagnoosia tehtäessä niiden mittaaminen potilailta on vakiotoimenpide.

Tiedon tiivistämisen menetelmät voidaan jakaa tiivistetystä datasta rekonstruoidun datan laadun perusteella häviöttömiin ja häviöllisiin menetelmiin. Häviöttömät menetelmät säilyttävät datan sellaisenaan ja häviölliset tuottavat jonkin approksimaation datasta, jossa pyritään säilyttämään sovelluksen kannalta tärkeät ominaisuudet.

Tutkimuksessa käsiteltiin tunnetuimmat häviöttömät menetelmät ja kehitettiin yksi uusi parametroituihin universaalikoodeihin perustuva menetelmä. Uusi menetelmä oli suorituskyvyltään samalla tasolla parhaiden tunnettujen menetelmien kanssa. Aiempiin biosignaaleiden tiivistyksestä saatuihin tuloksiin verrattuna tulokset olivat vertailukelpoisia ja joissain tapauksissa hieman parempiakin. Parhaiten toimivilla menetelmillä saatiin signaalien vaatima tila pienennettyä keskimäärin hieman alle puoleen alkuperäisestä.

Häviöllisistä menetelmistä sovellettiin tunnettuja skalaarikvantisointia, differentiaalista pulssikoodimodulaatiota ja muunnoskoodausta. Parhaat tiivistystulokset saavutettiin muunnoskoodauksella, jonka osana tarkasteltiin myös aallokkeita. Muunnoksen tiivistämisen menetelmistä testattiin kynnystämistä ja kehitettiin uusi kvantisointimenetelmä muunnoksen koodaamiseen. Uudella menetelmällä saavutettiin selvästi parempia tuloksia kuin kynnystämisellä.

2 Tutkimusongelma

Biosignaalilla tarkoitetaan tässä elävästä olennosta mitattua signaalia. Ihmisen tapauksessa tunnetuimmat ovat sydänsähkökäyrä, EKG, ja aivosähkökäyrä, EEG. Biosignaalien tiivistämistä on tutkittu paljon, mutta tutkimus on keskittynyt lähinnä EKG-signaalien tiivistämiseen. Sen tyypillisimmät sovellukset ovat signaalien arkistointi jatkotutkimusta varten ja tiedonsiirtokustannusten vähentäminen. Joissakin tapauksissa mittausten arkistointi saattaa olla myös lain vaatimaa. Jaladeddine *et al.* [18] esittävät yhteenvedon aiemmasta tutkimuksesta EKG:n tapauksessa ja Antoniol ja Tonella [2] käsittelevät EEG:n tiivistämistä.

Tässä tutkimuksessa kartoitettiin tiivitysmenetelmien toimivuutta uudentyyppisten biosignaalien tapauksessa. Tutkimusmateriaalina oli 69 Helsingin yliopistollisen keskussairaalan otoneurologisessa laboratoriossa mitattua silmänliikesignaalia. Signaaleita oli neljää eri tyyppiä: 22 sakaadia, 18 sinimuotoista signaalia, 15 vestibulo-okulaarista refleksiä ja 14 nystagmusta.

Signaalit mitataan kolmella tavallisella ihoelektrodilla potilaan ohimoilta ja otsalta. Sakaadin ja sinimuotoisen signaalin tapauksessa potilasta pyydetään seuraamaan seinälle ilmestyvää laservalotäplää katseellaan. Sakaadin tapauksessa täplä hyppii nopeasti vaakatasossa pisteestä toiseen neljän eri pisteen välillä. Sinimuotoisen signaalin tapauksessa piste liikkuu sulavasti vaakatasossa. Kummassakaan tapauksessa potilaan ei pitäisi olla mahdollista arvata pisteen liikettä, koska sakaadissa valotäplän liikkumisen ajat ja kohteet on satunnaistettu ja sinimuotoiseen signaaliin on sekoitettu kaksi eri taajuista siniaaltoa. Vestibulo-okulaarisen refleksin mittaamiseksi potilasta pyydetään pitämään katseensa yhdessä paikassa ja kääntämään päätään sivulta toiselle kuullessaan piippauksen. Nystagmus saadaan aikaan laittamalla kylmää tai kuumaa ilmaa potilaan toiseen korvakäytävään. [19]

Esimerkkejä signaaleista on kuvissa 2.1-2.4. Jokaisessa kuvassa on kymmenen sekunnin mittaisia pätkiä mittauksista; testiaineistossa mittausten pituus oli 20, 60 tai 80 sekuntia. Kuvissa signaalin liike ylöspäin vastaa silmien liikettä oikealle ja liike alaspäin silmien liikettä vasemmalle; vaaka-akseli on aika ja pystyakseli katseen kulma vaakasuunnassa. Kuvassa 2.1 on sakaadimittauksia; kuvan alin signaali on tutkimuksen kohinaisin. Kuvassa 2.2 on sinimuotoisia signaaleja, joissa on vain vähän kohinaa. Kuvassa 2.3 on vestibulo-okulaarisen refleksisignaalin mittauksia. Mitattaessa potilas saattaa räpyttää silmiään, joka aiheuttaa häiriöitä; kuvan 2.3 toisessa signaalissa on käynyt juuri näin. Piikit mittauksessa ovat silmänräpytyksiä; potilas on luultavasti räpyttänyt silmiään kuullessaan äänimerkin. Kuvan 2.4 sahalaitaiset

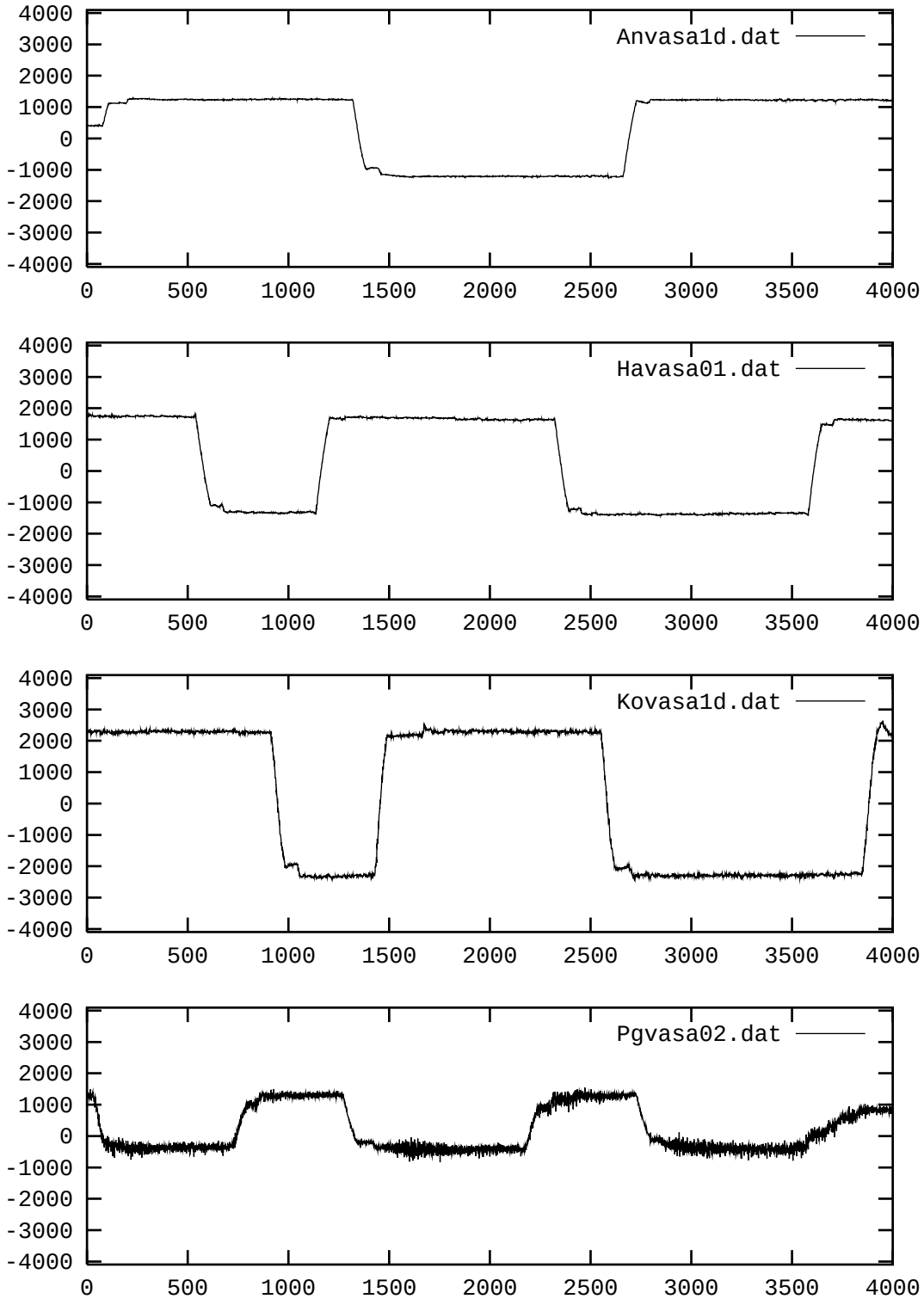
signaalit ovat nystagmusmittauksia.

Silmänliikesignaaleita käytetään apuna tasapainosairauksien diagnoosissa. Tunnetuin näistä on luultavasti Menieren tauti. Diagnoosia varten signaaleista mitataan mm. silmänliikkeiden nopeutta ja tarkkuutta. [19]

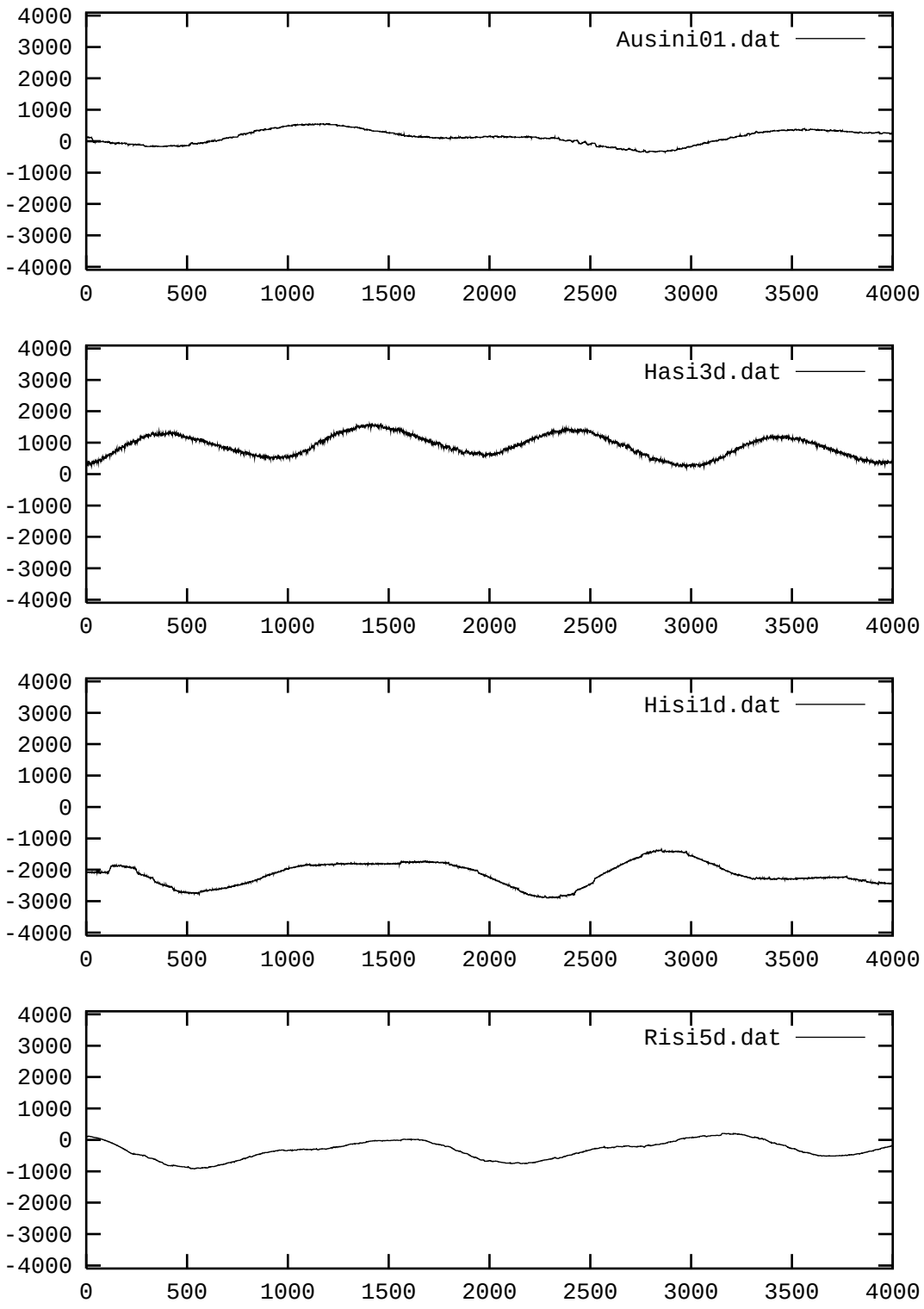
Signaalit on näytteistetty 400 Hz:n näytteenottotaajuudella ja näytteenottoresoluutiona on käytetty 13 bittiä. Arvioitu signaali-kohina-suhde on noin 32 dB. Häiriötä mittauksiin aiheuttavat pääasiassa kasvojen lihasten sähköinen toiminta ja verkkovirta. Verkkovirran aiheuttama häiriö näkyy mitatun signaalin spektrissä 50 Hz:n ja sen kanssa harmonisessa suhteessa olevien taajuuksien kohdalla. Sitä ei voida suodattaa pois, koska osa tarvittavasta informaatiosta on samalla kaistalla. Osa korkeamman taajuuden kohinasta suodatetaan yleensä pois ennen analyysiä. Tässä tutkimuksessa tutkittiin sekä suodatettuja että suodattamattomia signaaleita. Suodatukseen käytettiin muotoa

$$y(n) = \mathrm{Med}\left\{ \frac{1}{N}\sum_{i=1}^{N} x(n-i),\ x(n),\ \frac{1}{N}\sum_{i=1}^{N} x(n+i) \right\} \tag{2.1}$$

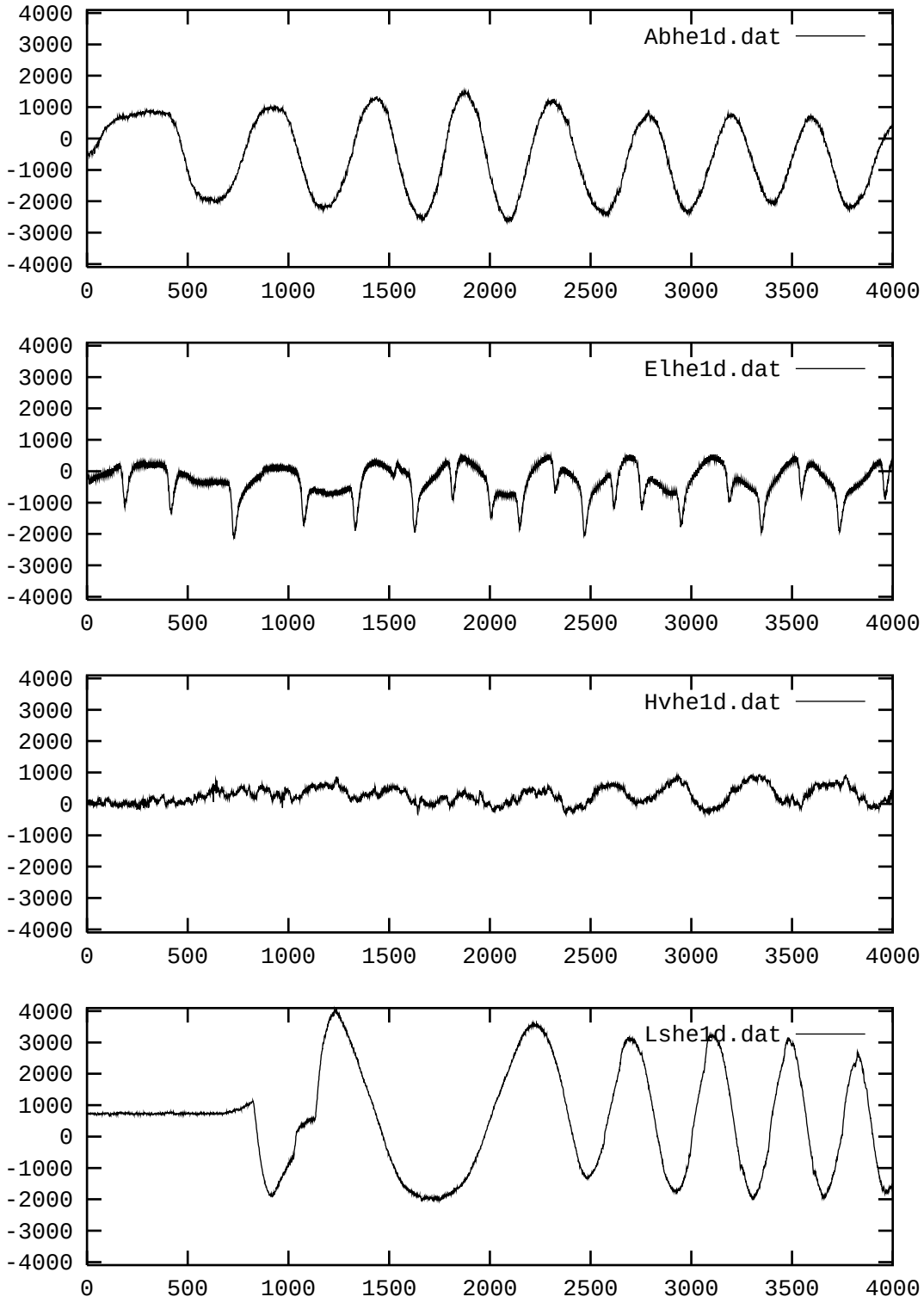
olevaa hybridimediaanisuodatinta, jonka on osoitettu säilyttävän signaalin lääketieteellisesti kiinnostavat parametrit hyvin [20, 21, 22]. Suodatuksen vaikutusta signaaleihin on havainnollistettu kuvassa 2.5.



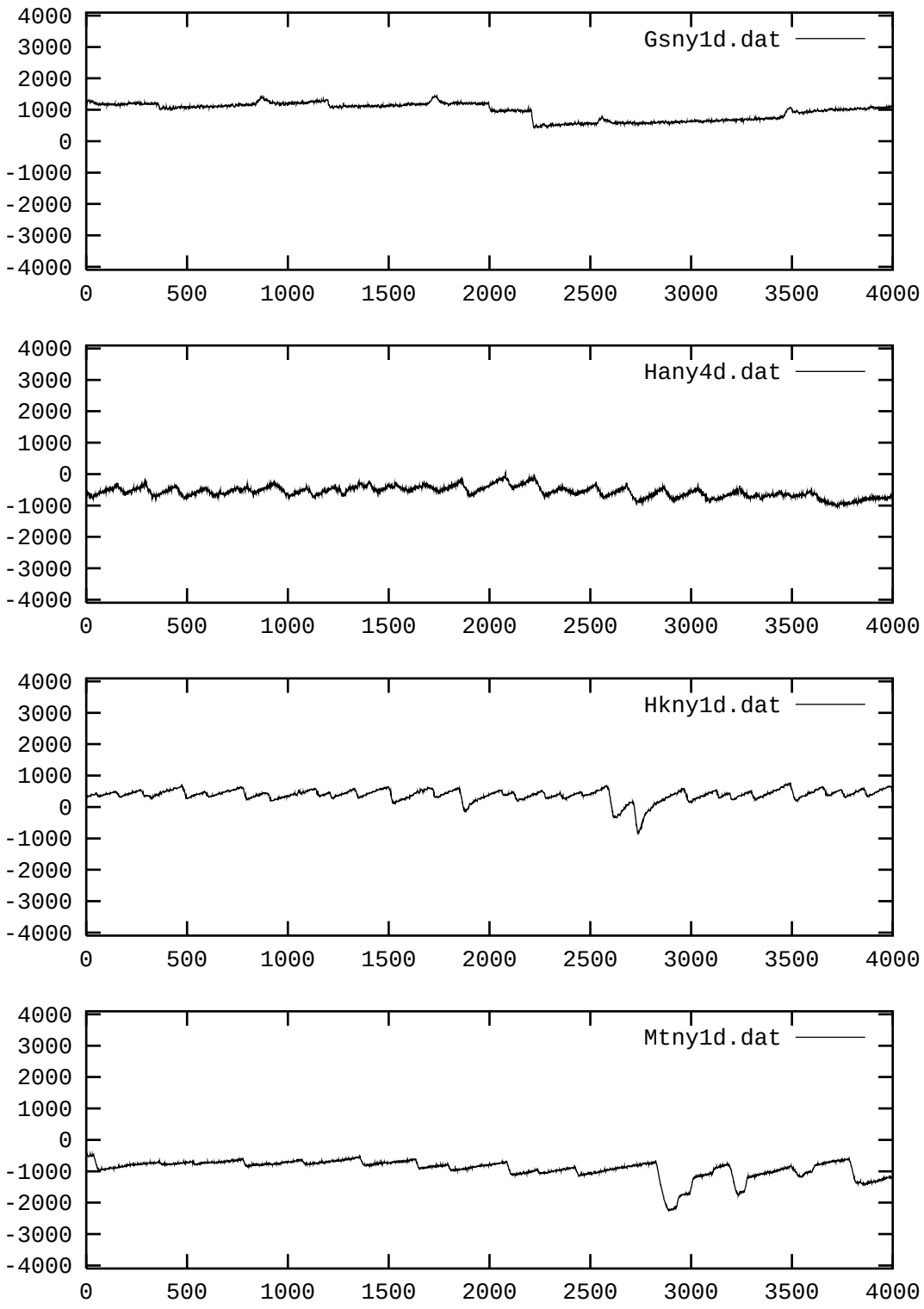
Kuva 2.1: Sakaadeja.



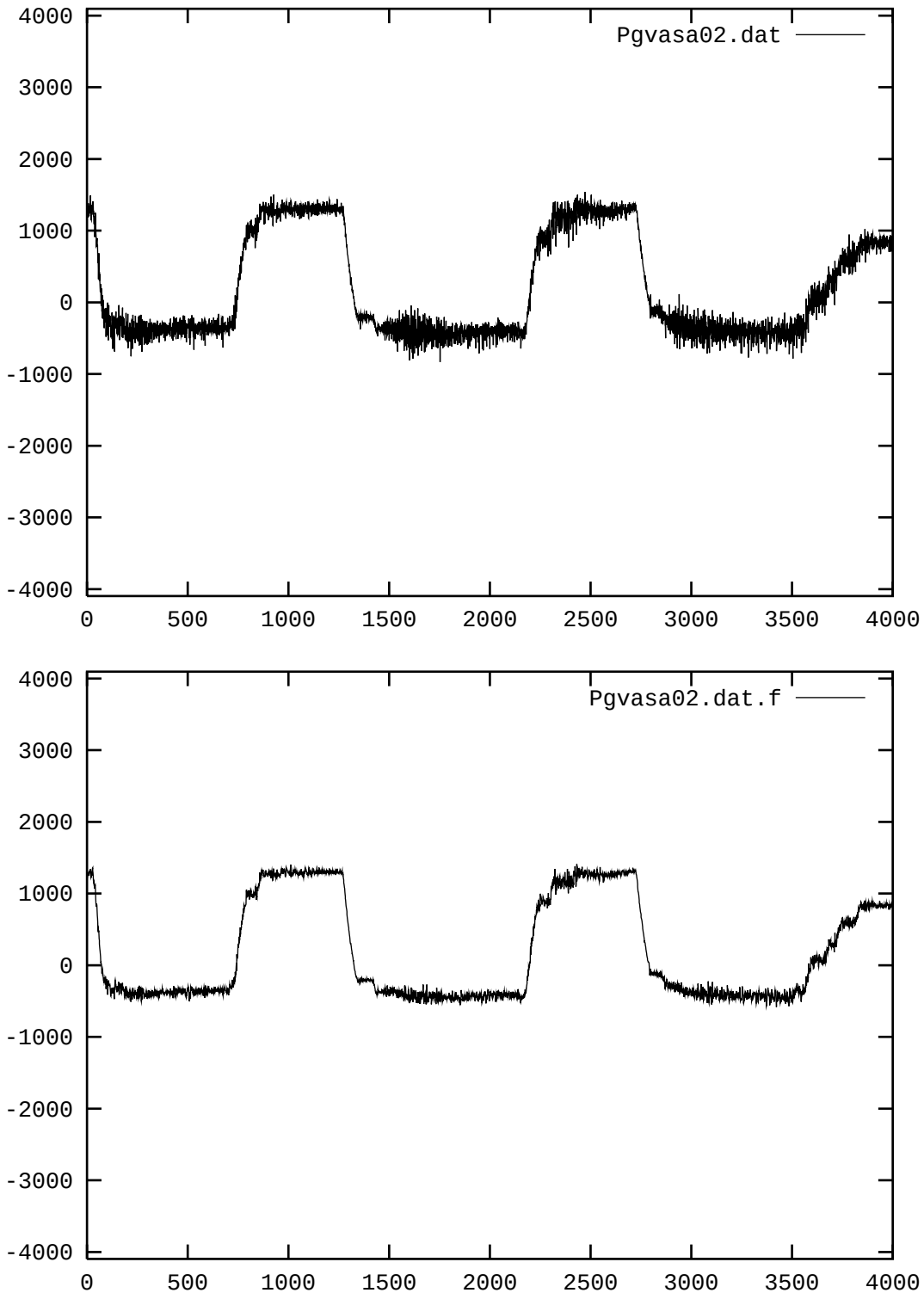
Kuva 2.2: Sinimuotoisia signaaleja.



Kuva 2.3: Vestibulo-okulaarisia refleksejä.



Kuva 2.4: Nystagmuksia.



Kuva 2.5: Alkuperäinen signaali ja signaali hybridimediaanisuodatuksen jälkeen.

3 Tiedon tiivistäminen

Tiedon tiivistämisen tavoitteena on esittää data D, joka koostuu jonkin aakkoston merkeistä, mahdollisimman tiiviissä muodossa. Data D koodataan pienempään muotoon $\Delta(D)$, joka on pystyttävä purkamaan takaisin alkuperäiseksi dataksi D tai joksikin riittävän hyväksi approksimaatioksi siitä. Tärkeimmät tiedon tiivistämisen edut ovat tiedontallennuskapasiteetin tarpeen pieneneminen ja tiedonsiirtokapasiteetin pienentynyt kulutus. [34]

Määritelmä 3.0.1 *Tiivistyssuhde* (compression ratio) CR mittaa tiivistyksen tehokkuutta. Se määritellään kaavalla

$$\mathrm{CR} = \frac{S_D}{S_{\Delta(D)}}, \tag{3.1}$$

missä S_D on datan koko jollakin yksiköllä mitattuna ja $S_{\Delta(D)}$ tiivistetyn datan koko samalla yksiköllä mitattuna.

Kaikkea tietoa ei voida tiivistää, mutta lähes kaikessa datassa on redundanssia, jota tiivistysalgoritmit voivat hyödyntää tiivistämisessä. Algoritmin tehokkuus riippuu paljon datan luonteesta. On todennäköistä, että jollekin datalle spesifi algoritmi tuottaa paremman tuloksen kuin geneerinen algoritmi. Teoreettisen pohjan tiedon tiivistämiselle antaa Shannonin *informaatioteoria* [33], jonka menetelmillä voidaan tutkia, kuinka paljon tietoa voidaan teoriassa tiivistää.

Tiedontiivistysalgoritmit voidaan jakaa kahteen luokkaan puretun datan laadun perusteella. *Häviöttömillä* (lossless, noiseless) algoritmeilla purettu data on identtinen alkuperäisen datan kanssa; *häviöllisillä* (lossy) algoritmeilla dekoodattu data on jokin approksimaatio alkuperäisestä datasta.[1] Toinen mahdollinen algoritmien jako on niiden toiminnan perusteella: *staattiset* algoritmit toimivat aina samoin ja *dynaamiset* mukautuvat datan paikallisiin vaihteluihin.

Häviöttömiä tiivistysmenetelmiä kutsutaan yleisnimikkeellä *entropian koodaus*. Entropian koodauksessa on tavoitteena vähentää lähetettyjen merkkien määrää siten, että alkuperäinen data pysyy muuttumattomana. Alkuperäisen datan säilyminen sellaisenaan on perusedellytys monille sovelluksille; esimerkiksi ohjelmien tiivistyksessä yhdenkin bitin virhe voi johtaa siihen, että ohjelmasta tulee käyttökelvoton. Entropian koodaajia käytetään myös häviöllisten menetelmien osana. Häviöllisten

[1] Häviöttömistä menetelmistä käytetään myös nimeä *säilyttävä* ja häviöllisistä nimeä *hukkaava*.

menetelmien tavoitteena on karsia sovelluksen kannalta merkityksetöntä informaatiota.

Entropian koodauksen menetelmistä tunnetuimpia ovat kokonaislukujen erilaisiin esityksiin perustuvat *universaalikoodit* [7, 38], *sanakirjamenetelmät* LZ77 [44] ja LZW [41], optimaalisen etuliitekoodin tuottava *Huffman-koodaus* ja viime aikoina paljon huomiota saanut informaatioteoreettisesti optimaalinen *aritmeettinen koodaus* [10, 16, 27, 31, 43]. Osa koodaajista ottaa huomioon merkkien keskinäisen järjestyksen säännönmukaisuudet ja yrittää mallintaa lähdettä koodauksen aikana.

Yleensä algoritmeista tarkastellaan tiivistyssuhteen lisäksi niiden tuottaman koodin virheensietokykyä ja viivettä, ts. tarvitseeko algoritmi useampia syötteen merkkejä koodin muodostamiseksi vai voiko se lähettää koodisanan heti, kun syötteen merkki tulee koodattavaksi. Algoritmin tehokkuus on myös tärkeää, varsinkin jos sovellus on reaaliaikainen (esim. musiikin tiivistäminen).

Tässä luvussa käsitellään perinteistä tiedon tiivistämisen teoriaa ja yleisiä häviöttömiä algoritmeja.

3.1 Aakkostot, merkit ja merkkijonot

Tiivistysalgoritmi kuvaa sanoman lähdeaakkostolta kohdeaakkostolle. Jotta sanoma voitaisiin rekonstruoida tiivistetystä tiedosta, on kuvauksen oltava injektio, ts. jokainen merkkijono kuvautuu vain yhdelle kohdeaakkoston merkkijonolle ja mitkään kaksi eri lähdeaakkoston merkkijonoa eivät kuvaudu samalle kohdeaakkoston merkkijonolle. [34]

Määritelmä 3.1.1 Aakkosto (alphabet) Σ on äärellinen joukko $\{a_0, a_1, \ldots, a_N\}$, joka sisältää ainakin yhden alkion. Aakkoston alkioita a_i kutsutaan merkeiksi.

Tyypillinen tiedon tiivistyksessä käytetty aakkosto on binäärinen aakkosto $\{0, 1\}$.

Määritelmä 3.1.2 Merkkijono (string) ω yli aakkoston Σ on jono aakkoston Σ merkkejä.

$$\omega = a_1 a_2 \ldots a_m,\ a_i \in \Sigma,\ i = 1, \ldots, m. \tag{3.2}$$

Merkkijonon ω pituutta merkitään $|\omega| = m$. Merkkijonojen $x = x_i$, $i = 1, \ldots, m$ ja $y = y_j$, $j = 1, \ldots, n$ katenointia merkitään:

$$xy = x_1 \ldots x_m y_1 \ldots y_n. \tag{3.3}$$

Katenoidun merkkijonon pituus on $|xy| = |x| + |y|$. Lisäksi määritellään tyhjä merkkijono ϵ, jossa ei ole yhtään merkkiä. Kaikkien aakkoston Σ merkkijonojen joukkoa, mukaanlukien ϵ, merkitään Σ^*.

Sanoma on äärellinen merkkijono yli lähdeaakkoston. Merkkijonoja vertaillaan toisiinsa etu- ja loppuliitteiden avulla.

Määritelmä 3.1.3 Merkkijono ω on merkkijonon x *etuliite* (prefix) $\omega \sqsubset x$, jos $x = \omega y$, jollekin $y \in \Sigma^*$.

Määritelmä 3.1.4 Merkkijono ω on merkkijonon x *loppuliite* (suffix) $\omega \sqsupset x$, jos $x = y\omega$, jollekin $y \in \Sigma^*$.

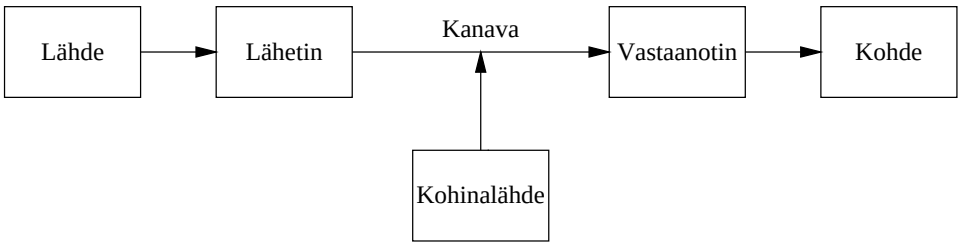
3.2 Informaatioteoriaa

Claude. E. Shannonin alunperin vuonna 1948 julkaisemasta kommunikaatiota koskevasta teoriasta [33] on muodostunut tiedon tiivistyksen teoreettinen perusta. Teoksessaan hän käsitteli yleistä kommunikaatiojärjestelmää (kuva 3.1), joka koostuu viidestä osasta:

- *Informaation lähde*, joka tuottaa *sanoman* tai sanomia, jotka välitetään kohteelle.
- *Lähetin*, joka muuttaa sanoman signaaliksi, joka voidaan lähettää kanavalla.
- *Kanava*, jota pitkin signaali välittyy lähettimestä vastaanottimeen. Kanava voi olla kohinainen.
- *Vastaanotin*, joka tekee lähettimeen nähden käänteisen operaation, eli konstruoi sanoman signaalista.
- *Kohde*, jolle sanoma on tarkoitettu.

Sanoma valitaan mahdollisten sanomien joukosta. Niillä voi olla merkitys; sanomat voivat viitata maailman olioihin tai käsitteisiin. Shannonin teoria ei kuitenkaan koske kommunikaation semanttista puolta.

Kommunikaation ongelmana on rekonstruoida jossain paikassa sanoma, joka on valittu muualla. Sanoma olisi pystyttävä lähettämään mahdollisimman taloudellisesti. [33]



Kuva 3.1: Yleinen kommunikaatiojärjestelmä.

Informaation lähde tuottaa sanoman merkki merkiltä. Merkit voidaan valita eri todennäköisyyksillä ja merkin valinta voi riippua edellisistä merkeistä. Yleensä puhutaan lähteen tuottamista *tapahtumista*.

Jos aiemmilla tapahtumilla ei ole vaikutusta seuraavaan tapahtumaan puhutaan 0. asteen lähteestä. Tällaisella lähteellä jokaisen tapahtuman todennäköisyys on aina sama eli tapahtumat ovat keskenään tilastollisesti riippumattomia. Tällaisen lähteen kuvaamiseen riittää sen todennäköisyysjakauma. Tilastollisesti riippumattomalle lähteelle on yhdistetylle tapahtumalle voimassa

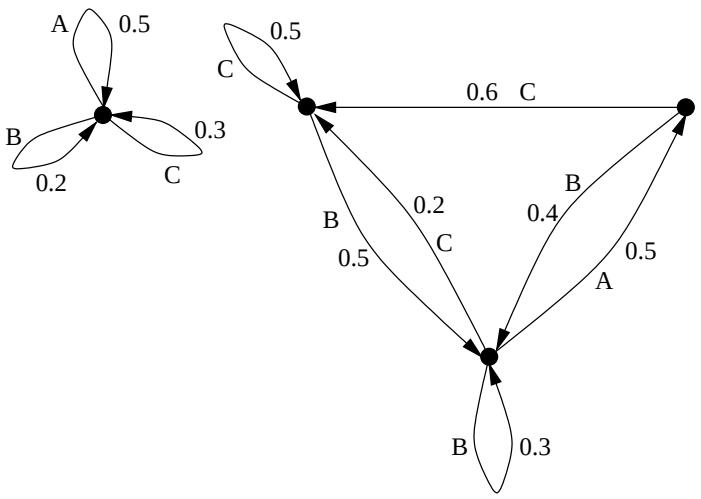
$$p(i,j) = p(i)p(j). \tag{3.4}$$

Kun tapahtumat ovat toisistaan tilastollisesti riippuvia käytetään usein lähteen mallina *Markovin prosessia* [33]. Markovin prosessilla on äärellinen joukko tiloja, joista voidaan siirtyä toisiin tiloihin eri todennäköisyyksillä; jokainen tilasiirtymä tuottaa tapahtuman. Esimerkiksi ensimmäisen asteen Markovin prosessilla tapahtuman todennäköisyys riippuu vain edellisestä tapahtumasta. Tällöin on voimassa

$$p(i,j) = p(j|i)p(i). \tag{3.5}$$

Kyseisessä mallissa tapahtumaketjun x_1, x_2, x_3, x_4 todennäköisyys voidaan laskea seuraavasti:

$$\begin{aligned} p(x_1,x_2,x_3,x_4) &= p(x_2,x_3,x_4|x_1)p(x_1) && (3.6)\\ &= p(x_3,x_4|x_2)p(x_2|x_1)p(x_1) && (3.7)\\ &= p(x_4|x_3)p(x_3|x_2)p(x_2|x_1)p(x_1). && (3.8) \end{aligned}$$



Kuva 3.2: Markovin prosesseja.

		Y		
	$p(Y\|X)$	A	B	C
	A	0	0.4	0.6
X	B	0.5	0.3	0.2
	C	0	0.5	0.5

Taulukko 3.1: Kuvan 3.2 oikeanpuoleisen graafin matriisi.

Esimerkki 3.2.1 Markovin prosesseja havainnollistetaan usein graafien avulla. Kuvassa 3.2 on kuvattu kaksi erilaista Markovin prosessia. Vasemmalla oleva graafi kuvaa 0. asteen ja oikealla oleva 1. asteen Markovin prosessia. Vasen graafi vastaa lähdettä, jonka tapahtumien todennäköisyydet ovat $p(X = A) = 0.5$, $p(X = B) = 0.2$ ja $p(X = C) = 0.3$. Oikeanpuoleinen graafi voidaan kuvata ehdollisten todennäköisyyksien taulukkona (taulukko 3.1).

Sanoman sisältämä informaatio riippuu sen todennäköisyydestä mahdollisten sanomien joukossa. Sanomista käytetään myös nimitystä *tapahtuma*.

Määritelmä 3.2.1 Jos X on tapahtuma, jonka mahdolliset vaihtoehdot ovat $\{x_i\}$, niin tapahtuman $X = x_i$ *informaatio* I on

$$I(X = x_i) = -\log_k p(X = x_i), \tag{3.9}$$

missä k voidaan valita halutun yksikön mukaan. Kun $k = 2$ informaation mittayksikkö on bitti, kun $k = 10$ mittayksikkö on desimaaliluku. Joskus kantalukuna on e,

jolloin puhutaan *luonnollisista yksiköistä* (natural units tai "nats"). [33]

Esimerkiksi yksi desimaaliluku, joka voi saada kymmenen eri arvoa ja jokaisen samalla todennäköisyydellä, sisältää noin 3.32 bittiä informaatiota. [33]

Oletetaan, että tiedetään vain tapahtumien todennäköisyydet p_i. *Entropia* mittaa tällaiseen tilanteeseen liittyvää epävarmuutta tai valintaa. Entropian mitan $H(p_1, p_2, \ldots, p_n)$ on täytettävä seuraavat vaatimukset:

1. H on jatkuva.

2. Jos kaikki p_i ovat yhtäsuuria, $p_i = 1/n$, on H monotonisesti kasvava n:n funktio.

3. Jos valinta jaetaan osiin, niin yhdistetty entropia on osien entropian painotettu summa.

   Jaetaan tapahtumat $p_i = \{1/2, 1/3, 1/6\}$ kahteen osaan $\{1/2\}$ ja $\{1/3, 1/6\}$. Tällöin

$$H\left(\frac{1}{2}, \frac{1}{3}, \frac{1}{6}\right) = H\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{1}{2}H(1) + \frac{1}{2}H\left(\frac{2}{3}, \frac{1}{3}\right). \tag{3.10}$$

Shannonin [33] mukaan ainoa ehdot toteuttava funktio on muotoa

$$H = -K\sum_{i=1}^{n} p_i \log p_i, \tag{3.11}$$

missä K on positiivinen vakio, joka valitaan tässä ykköseksi. Entropia H on nolla, jos ja vain jos yhden tapahtuman todennäköisyys on yksi (muiden todennäköisyys on tällöin nolla). Intuitiivisesti tämä on selvää, koska varma lopputulos ei sisällä epävarmuutta. Jos tapahtumia on n kappaletta, niin maksimissaan entropia on $\log n$, kun kaikkien tapahtumien todennäköisyydet ovat yhtä suuria. Tällöin tilanteeseen liittyy eniten epävarmuutta. Jokainen muutos, joka tasoittaa lähteen todennäköisyysjakaumaa lisää entropiaa.

Määritelmä 3.2.2 Jos X on tapahtuma, jonka vaihtoehdot ovat $\{x_i\}$, ja tapahtumat eivät riipu edellisistä tapahtumista, niin X:n entropia on

$$H(X) = -\sum_{i} p(X = x_i) \log p(X = x_i), \tag{3.12}$$

missä logaritmin kantaluku valitaan mittayksikön mukaan. [33]

Määritelmä 3.2.3 Olkoot X ja Y tapahtumia, joista X:n vaihtoehdot ovat $\{x_i\}$ ja Y:n vaihtoehdot $\{y_j\}$. Yhdistetyn tapahtuman X, Y entropia on tällöin

$$H(X,Y) = -\sum_{i,j} p(X = x_i, Y = y_j) \log p(X = x_i, Y = y_j). \tag{3.13}$$

Tapahtumien X ja Y entropia saadaan

$$H(X) = -\sum_{i,j} p(X = x_i, Y = y_j) \log \sum_{j} p(X = x_i, Y = y_j), \tag{3.14}$$

$$H(Y) = -\sum_{i,j} p(X = x_i, Y = y_j) \log \sum_{i} p(X = x_i, Y = y_j). \tag{3.15}$$

Voidaan osoittaa, että

$$H(X,Y) \leq H(X) + H(Y). \tag{3.16}$$

Yhtäsuuruus on voimassa vain silloin, kun tapahtumat X ja Y ovat riippumattomia eli $p(X = x_i, Y = y_j) = p(X = x_i)p(Y = y_j)$. [33]

Määritelmä 3.2.4 Jos kaksi tapahtumaa eivät ole riippumattomia, jolloin todennäköisyys Y:n arvolle y_j, jos X:n arvo tiedetään x_i:ksi, on

$$p(Y = y_j | X = x_i) = \frac{p(X = x_i, Y = y_j)}{\sum_j p(X = x_i, Y = y_j)}. \tag{3.17}$$

Tapahtuman Y *ehdollinen entropia*, jos X tunnetaan, $H(Y|X)$ määritellään painotettuna keskiarvona Y:n entropioista kaikilla eri X:n arvoilla. Painotuksena käytetään yhdistetyn tapahtuman $X = x_i, Y = y_j$ todennäköisyyttä eli

$$H(Y|X) = -\sum_{i,j} p(X = x_i, Y = y_j) \log p(Y = y_j | X = x_i) = H(X,Y) - H(X). \tag{3.18}$$

Ehdolliselle entropialle voidaan osoittaa, että on aina voimassa

$$H(Y) \geq H(Y|X). \tag{3.19}$$

Siis tieto X:stä ei koskaan lisää epävarmuutta. [33]

Markovin prosessin entropia voidaan laskea kaavasta

$$H = \sum_i P_i H_i = -\sum_{i,j} P_i p(Y = y_j | S = s_i) \log p(Y = y_j | S = s_i), \tag{3.20}$$

missä P_i on tilan $S = s_i$ todennäköisyys, H_i tilan $S = s_i$ entropia ja Y on tapahtuma, jonka mahdollisuudet ovat kaikki tilasiirtymät tilasta s_i. Markovin prosessin entropia saadaan siis laskemalla painotettu keskiarvo tilojen entropiasta käyttäen painotuksena tilan todennäköisyyttä. [33]

Data koostuu informaatiosta ja *redundanssista*. Redundanssi on nimensä mukaisesti toistoa, joka voidaan eliminoida datasta ja myöhemmin lisätä takaisin datan muuttumatta. Informaatiota ei voida poistaa datasta muuttamatta sitä. Häviöttömät tiivistysmenetelmät ovat redundanssia vähentäviä; ne eivät siis vähennä informaatiota. Häviölliset menetelmät vähentävät sekä redundanssia että informaatiota. [25]

Määritelmä 3.2.5 Lähteen, joka voi tuottaa n tapahtumaa, redundanssi R on

$$R = \log n - H, \tag{3.21}$$

missä H on lähteen entropia. [33]

Redundanssi on siis maksimientropian ja todellisen entropian erotus. Lähteellä ei ole redundanssia kun entropia on maksimissaan eli $H = \log n$.

Määritelmä 3.2.6 Lähteen *suhteellinen redundanssi* Z on lähteen redundanssin ja maksimientropian suhde:

$$Z = \frac{R}{\log n} = 1 - \frac{H}{\log n}. \tag{3.22}$$

Lynchin [25] mukaan entropian avulla saadaan tiivistyssuhteen CR ylärajaksi

$$\mathrm{CR}_{max} = \frac{\log n}{H}. \tag{3.23}$$

Erilaisia lähteen malleja, yleensä tilastollisesti riippumaton 0. asteen lähdettä ja 1. asteen Markovin prosessia, käytetään tiivistysmenetelmien testaamiseen. [25]

Maksimaalisen häviöttömän tiivistyksen saavuttamiseksi voidaan lähteen käyttäytyminen muuttaa Markovin prosessi-mallista tilastollisesti riippumattomaksi malliksi ja koodata optimaalisesti riippumattoman mallin tuottamat tapahtumat. Op-

timaalisella koodauksella koodisanojen keskipituus ℓ on sama kuin lähteen entropia H. [25]

Informaatioteorian mukaan entropia on siis määritelty suhteessa tilastolliseen malliin. Usein lähteen todellista mallia ei tunneta, joten se joudutaan jollain menetelmällä konstruoimaan empiirisestä aineistosta. Mitä parempi lähteen malli on, sitä pienempi on lähteen entropia suhteessa malliin; toisaalta monimutkainen malli vie enemmän tilaa tiedostosta.

3.3 Universaalikoodaus

Universaalikoodauksessa ei tarvitse tuntea lähteen jakaumaa, vaan riittää että voidaan järjestää tapahtumat todennäköisyyksien mukaan laskevaan järjestykseen. Koodauksessa tapahtuman järjestysnumero koodataan jollain yksinkertaisella koodilla. Universaalikoodauksen tuottamien koodisanojen keskipituudelle ℓ on voimassa

$$\ell \leq c_1 H + c_2, \tag{3.24}$$

missä H on lähteen entropia ja c_1 ja c_2 ovat vakioita. Koodia sanotaan asymptoottisesti optimaaliseksi, jos $c_1 = 1$. Silloin koodisanojen keskipituus ℓ lähestyy entropiaa entropian kasvaessa. Tässä esitetyt koodit ovat *staattisia* ja yksiselitteisesti purettavia. Staattisuudella tarkoitetaan sitä, että samalle merkille tuotetaan aina sama esitys. Monien koodien yksikäsitteisyys perustuu siihen, että koodisanan pituus saadaan jotenkin selville siitä itsestään. Esimerkiksi koodin alussa voi olla sarja ykkösiä ja niiden jälkeen nolla; tämän perusteella voidaan päätellä koodin loppuosan pituus. [7]

Koodien toteutus on laskennallisesti tehokasta ja varsinkin parametroituja universaaleja koodeja soveltamalla saadaan hyviä tiivistystuloksia. Esimerkiksi Howardin ja Vitterin progressiivisessa FELICS-kuvantiivistysalgoritmissa [17] on käytetty Golomb-, Rice- ja Subexponential-koodeja, joille yritetään ennustaa paras parametrin arvo.

3.3.1 γ-koodi

Elias [7] esittää koodeja, jotka sopivat epätasaisesti jakautuneille kokonaisluvuille. Tunnettu γ-koodi toimii hyvin luvuille, jotka ovat jakautuneet eksponentiaalisesti

siten, että 0 on todennäköisin. Luvun x koodi $\gamma(x)$ muodostetaan seuraavasti:

$$\gamma(x) = \overbrace{1\ldots1}^{n}0\overbrace{x_{n-1}\ldots x_1}^{x'}, \tag{3.25}$$

missä $n = 0$, kun $x = 0$, ja $n = \lfloor \log_2 x \rfloor + 1$, kun $x > 0$, ja x' luvun x binääriesitys ilman ylintä 1-bittiä. Saatu koodi on lyhyempi pienille kokonaisluvuille ja pidempi suurille. Taulukossa 3.2 on esitetty γ-koodit luvuille $0 \ldots 14$; pisteellä osoitetaan koodin loogisten osien välit.

x	$\gamma(x)$	x	$\gamma(x)$	x	$\gamma(x)$
0	0	5	111.0.01	10	1111.0.010
1	1.0	6	111.0.10	11	1111.0.011
2	11.0.0	7	111.0.11	12	1111.0.100
3	11.0.1	8	1111.0.000	13	1111.0.101
4	111.0.00	9	1111.0.001	14	1111.0.110

Taulukko 3.2: Lukujen $0 \ldots 14$ γ-koodit.

3.3.2 δ-koodi

Toinen Eliasin koodi, δ, on asymptoottisesti optimaalinen; se muodostetaan koodaamalla ensin luvun binääriesityksen pituus γ-koodilla ja sen jälkeen luvun binääriesitys ilman ylintä 1-bittiä:

$$\delta(x) = \overbrace{\gamma_n \ldots \gamma_1}^{\gamma(\lfloor \log_2(x) \rfloor + 1)} \overbrace{x_k \ldots x_1}^{x'}, \tag{3.26}$$

missä x' on luvun x binääriesitys ilman ylintä 1-bittiä. [7]

3.3.3 Golomb-koodi

Golomb-koodi on saanut nimensä kehittäjältään S. W. Golombilta. Howardin ja Vitterin [17] mukaan se toimii hyvin eksponentiaalisille jakaumille; lisäksi sitä voidaan säätää parametrin avulla sopiviksi erilaisille jakaumille. Golomb-koodi $Gol(x, k)$ parametrilla k luvulle x muodostetaan seuraavasti:

$$Gol(x, k) = \overbrace{1\ldots1}^{n}0\overbrace{y_j \ldots y_1}^{y}, \tag{3.27}$$

missä $n = \lfloor x/k \rfloor$ ja y on luvun x mod k esitys tavalla, jossa säästetään bittejä, kun se on mahdollista. Koska x:n jakojäännös k:n suhteen on aina k:ta pienempi, voidaan useimmissa tapauksissa käyttää lukujen $0, \ldots, k-1$ esittämiseen vaihtelevanmittaista koodia, jolla koodisanojen keskipituus on pienempi kuin pienimmällä kiinteällä bittien määrällä saavutettava $\lceil \log_2 k \rceil$.

3.3.4 Rice-koodi

Howardin ja Vitterin [17] mukaan R.F. Ricen kehittämä Rice-koodi toimii myös hyvin eksponentiaalisesti jakautuneille luvuille. Sen etuna γ-koodiin on se, ettei koodin pituus kasva yhtä nopeasti. Rice-koodi on parametroitu, joten se voidaan sovittaa lähteen jakaumalle sopivaksi. Itseasiassa Rice-koodi on Golomb-koodin erikoistapaus, jossa Golomb-koodin parametri on rajattu kahden potenssiksi. Luvun x Rice-koodi $Rice(x, k)$ parametrilla k $(k \geq 0)$ saadaan seuraavasti:

$$Rice(x, k) = \overbrace{1 \ldots 1}^{m} 0 \overbrace{y_k \ldots y_1}^{y}, \tag{3.28}$$

missä $m = \lfloor x/2^k \rfloor$ ja y on luvun x mod 2^k binääriesitys k bitillä. Rice-koodi on Golomb-koodia yksinkertaisempi siinä, että etuosan 1-bittien jälkeen on aina vakiomäärä bittejä; toisaalta Golomb-koodissa on tiheämpi parametrointi, joten sitä voidaan säätää tarkemmin. Taulukossa 3.3 on esitettynä Rice-koodeja eri parametrin arvoilla.

x	$Rice(x,0)$	$Rice(x,1)$	$Rice(x,2)$	$Rice(x,3)$
0	0	0.0	0.00	0.000
1	1.0	0.1	0.01	0.001
2	11.0	1.0.0	0.10	0.010
3	111.0	1.0.1	0.11	0.011
4	1111.0	11.0.0	1.0.00	0.100
5	11111.0	11.0.1	1.0.01	0.101
6	111111.0	111.0.0	1.0.10	0.110
7	1111111.0	111.0.1	1.0.11	0.111
8	11111111.0	1111.0.0	11.0.00	1.0.000

Taulukko 3.3: Rice-koodeja luvuille $0 \ldots 8$.

3.3.5 Eksponentiaalinen Golomb-koodi

Jukka Teuholan kehittämä eksponentiaalinen Golomb-koodi toimii hyvin eksponentiaalisilla jakaumilla. Eksponentiaalinen Golomb-koodi muistuttaa läheisesti Golomb-koodia; siinä on vain käytetty eksponentiaalisesti kasvavia jakajia. Koodi $exp\text{-}Gol(x,k)$ parametrilla k muodostetaan seuraavasti:

$$exp\text{-}Gol(x,k) = \overbrace{1\ldots1}^{m}0\overbrace{y_{m-1}\ldots y_1}^{x'}, \tag{3.29}$$

missä $m = \lfloor \log_2 x \rfloor - k$, jos $\lfloor \log_2 x \rfloor \geq k$ tai 0, jos $\lfloor \log_2 x \rfloor < k$ ja x' on luvun x binääriesityksen $k + m + 1$ alinta bittiä. Eksponentiaalinen Golomb-koodi on keskimääräisessä tapauksessa hieman huonompi kuin Golomb-koodi, mutta jos koodin parametria ennustetaan, niin ennustuksessa tehty virhe ei aiheuta suurta tappiota tiivistyksessä. Eksponentiaalisia Golomb-koodeja on esitetty taulukossa 3.4. [38]

Alunperin Golomb- ja eksponentiaalinen Golomb-koodi kehitettiin bittivektorien *juoksunpituuden tiivistykseen*; koodilla ilmoitetaan, montako peräkkäistä nollaa tai ykköstä on syötteessä. Bittivektorien tiivistyksellä on monia käytännön sovelluksia, kuten mustavalkokuvien tiivistäminen.

x	$exp\text{-}Gol(x,0)$	$exp\text{-}Gol(x,1)$	$exp\text{-}Gol(x,2)$	$exp\text{-}Gol(x,3)$
0	0	0.0	0.00	0.000
1	1.0.0	0.1	0.01	0.001
2	1.0.1	1.0.00	0.10	0.010
3	11.0.00	1.0.01	0.11	0.011
4	11.0.01	1.0.10	1.0.000	0.100
5	11.0.10	1.0.11	1.0.001	0.101
6	11.0.11	11.0.000	1.0.010	0.110
7	111.0.000	11.0.001	1.0.011	0.111
8	111.0.001	11.0.010	1.0.100	1.0.0000

Taulukko 3.4: Eksponentiaalisia Golomb-koodeja luvuille $0 \ldots 8$.

3.3.6 Etumerkilliset luvut

Edellä käsitellyt koodit toimivat vain positiivisille kokonaisluvuille. Usein joudutaan koodaamaan myös negatiivisia lukuja. Etumerkillisten lukujen käsittelyyn on useita tapoja, joista voidaan valita paras erilaisille jakaumille.

Yksinkertaisin tapa on lisätä koodin perään yksi bitti ilmoittamaan luvun etumerkki; tällöin saadaan yhtä pitkät koodit luvuille x ja $-x$. Jos luku on nolla, ei merkkibittiä lisätä. Tällä menettelyllä nollan koodi on lyhyempi kuin muiden.

Negatiiviset luvut voidaan limittää positiivisten sekaan. Koodataan $2x - 1$, kun $x > 0$, ja $-2x$, kun $x \leq 0$, eli positiiviset luvut esitetään parittomilla luvuilla ja negatiiviset parillisilla, näin saadaan positiivisille luvuille lyhyempi koodi kuin negatiivisille. Sama voidaan tehdä myös toisinpäin, jolloin negatiivisten lukujen koodit ovat hieman lyhyempiä kuin positiivisten.

Muita menetelmiä on kirjallisuudessa käsitelty tarkemmin erityisesti häviöttömän kuvantiivistyksen yhteydessä, jossa merkinkäsittelyn eri tavat on otettu mukaan koodin mukautumisstrategioihin.

3.3.7 Mukautuvat universaalit koodit

Kehitin yksinkertaisen menetelmän, jolla voidaan tehostaa tiivistystä, jos lähteen merkkien jakaumassa on paikallisia eroja. Menetelmän ideana on jakaa syöte lohkoihin ja koodata kukin lohko optimaalisella koodilla. Käytössä on aina yksi parametroitu koodi.

Algoritmissa 1 esitetty menetelmä on helppo toteuttaa. Optimaalisen parametrin etsimistä voidaan tehostaa huomaamalla, että puskurin koodatun esityksen pituus laskee monotonisesti kohti minimiä, kun käytetään esimerkiksi eksponentiaalinen Golomb- tai Rice-koodia. Tällöin voidaan aloittaa minimipituuden etsintä edellisestä optimaalisesta parametrista ja edetä aina suuntaan, jossa puskurin koodatun esityksen pituus on pienempi. Vaikka kokonaisuutena menetelmä on melko yksinkertainen, en löytänyt siitä mainintaa kirjallisuudesta.

Algoritmi 1 Mukautuva universaalikoodaus

1: **while** syötettä jäljellä **do**
2: Lue puskuri täyteen syötettä. Jos syöte loppuu kesken, voidaan täyttää puskuri vain osittain.
3: **for all** järkevät koodin pituudet k **do**
4: Laske puskurin koodiesityksen pituus, kun se koodataan käyttäen parametria k.
5: **end for**
6: Valitse k_{opt}, jolla puskurin koodattu esityksen pituus on lyhin.
7: Koodaa tieto parametrista ja puskurin sisältö käyttäen parametria k_{opt}.
8: **end while**

3.4 Juoksunpituuden koodaus

Juoksunpituuden koodaus (run-length coding, RLE) perustuu usein toistuvien merkkien peräkkäisten juoksujen pituuksien laskentaan ja juoksujen korvaamiseen niiden pituudella. Juoksujen pituudet koodataan sopivalla koodilla (esimerkiksi joku universaalikoodi tai Huffman-koodi). RLE-koodaus on ollut käytössä mm. LBM-kuvaformaatissa ja faxeissa, joissa yhden värin muodostamat tasaiset alueet tiivistyvät hyvin RLE:llä. Sellaisenaan RLE:n käyttö ei ole kovin yleistä, mutta sitä voidaan monissa tapauksissa soveltaa osana tiivistysjärjestelmää.

Esimerkki 3.4.1 Merkkijono *aaabccbbba* voidaan koodata muodossa (3,a), (1,b), (2,c), (3,b), (1,a).

3.5 Shannon-Fano-koodaus

Kun lähteen jakauma tunnetaan tarkoin, voidaan lähteelle kehittää universaalikoodeja tehokkaampia koodeja. Ongelmana on siis löytää sellainen koodisanojen joukko, jolla ℓ on mahdollisimman pieni, kun lähteen tapahtumien todennäköisyydet tunnetaan. Shannon ja Fano julkaisivat toisistaan riippumatta algoritmit ongelman ratkaisemiseksi. Ratkaisu ei kuitenkaan ole paras mahdollinen. Algoritmit poikkesivat hieman toisistaan; Shannonin algoritmi perustui kumulatiivisten todennäköisyyksien binääriesityksiin ja Fanon joukkoihin jakamiseen. Lopputuloksena saadun koodin tehokkuus on kuitenkin molemmissa tapauksissa sama. Yksinkertaisuuden vuoksi esitän vain Fanon algoritmin (Algoritmi 2), joka ratkaisee ongelman osittavalla menetelmällä. [33]

Algoritmi 2 Fanon algoritmi, FANO(Σ, ω)

Alkuehto: Σ on lähdeaakkosto. Merkkien todennäköisyydet tunnetaan ja ensimmäisellä kutsulla ω:n arvo on ϵ.

Jättöehto: Kaikkien lähdeaakkoston merkkien a_i koodisanat ω on määrätty.

1: **if** $|\Sigma| > 1$ **then**
2: Jaa aakkosto Σ kahteen todennäköisyyksiltään suurinpiirtein yhtä suureen joukkoon Σ_1 ja Σ_2.
3: FANO(Σ_1, $\omega 0$).
4: FANO(Σ_2, $\omega 1$).
5: **else**
6: Aseta merkin $a \in \Sigma$ koodisanaksi ω.
7: **end if**

3.6 Huffman-koodaus

D. A. Huffman kehitti vuonna 1952 kokoavan menetelmän, jolla voidaan muodostaa optimaalinen etuliitekoodi, kun tunnetaan lähteen todennäköisyysjakauma. Jos lähteen merkkien todennäköisyydet ovat 1/2:n potensseja on Huffmanin algoritmin tuottama koodi optimaalinen myös informaatioteoreettisessa mielessä. [11]

Teoreema 3.6.1 Optimaalisella binäärisellä etuliitekoodilla on seuraavat ominaisuudet:

1. Olkoot merkkien a ja b koodisanojen pituudet $\ell(a)$ ja $\ell(b)$ ja todennäköisyydet $p(a)$ ja $p(b)$. Jos $p(a) > p(b)$, niin $\ell(a) \leq \ell(b)$. Todennäköisillä merkeillä on siis lyhyemmät koodisanat kuin epätodennäköisillä.

2. Kahden epätodennäköisimmän merkin koodisanat, jotka ovat yhtä pitkiä, eroavat toisistaan vain koodisanan viimeisen merkin osalta.

Todistus: ks. [11, s. 270]

Gershon ja Grayn [11] mukaan Huffmanin algoritmi (Algoritmi 3) tuottaa ahneella menetelmällä koodipuun, joka täyttää teoreeman 3.6.1 mukaiset optimaaliseen binäärisen etuliitekoodin vaatimukset. Itseasiassa teoreema 3.6.1 antaa tarvittavat tiedot optimaalisen koodin konstruointiin. Huffmanin algoritmi yhdistää aakkoston Σ kaksi epätodennäköisintä merkkiä a_1 ja a_2 uudeksi merkiksi $a_{1,2}$ ja muodostaa uuden aakkoston $\Sigma' = (\Sigma - \{a_1, a_2\}) \cup \{a_{1,2}\}$. Näin jatketaan, kunnes aakkostossa on enää yksi merkki; koodi saadaan merkkien yhdistämisistä siten, että jokainen merkkien yhdistäminen muodostaa yhden koodipuun solmun (ks. kuva 3.3). Huffmanin algoritmi voidaan toteuttaa aikavaatimuksella $O(n \log n)$.

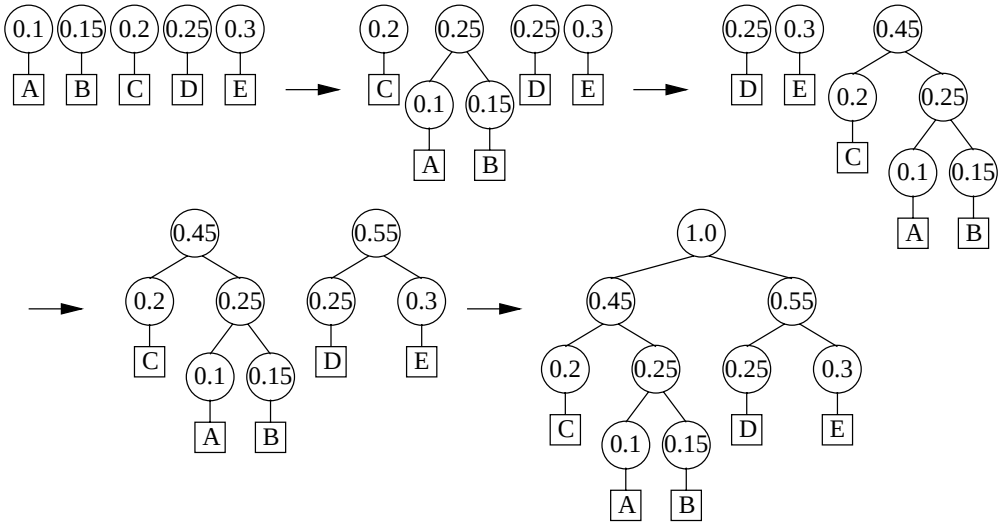
Algoritmin 3 muodostamaa koodipuuta käytetään koodaukseen siten, että koodisana muodostetaan reitistä juuresta merkkiä vastaavaan lehtisolmuun. Vasemman alipuun valintaa merkitään 0-bitillä ja oikean 1-bitillä. Koodi on aina yksikäsitteisesti purettavissa, koska yhdelläkään solmulla, johon liittyy merkki, ei ole alipuita. Koodi puretaan siten, että aloitetaan juuresta, luetaan bittejä ja edetään koodipuussa alaspäin kunnes tullaan solmuun, johon liittyy merkki. Tulostetaan merkki ja siirrytään takaisin juureen.

Esimerkki 3.6.1 Kuvassa 3.3 on esitetty koodin muodostaminen aakkostolle $\{A, B, C, D, E\}$, jonka merkkien todennäköisyydet (samassa järjestyksessä) ovat 0.1,

Algoritmi 3 Huffmanin algoritmi

Alkuehto: Σ on lähdeaakkosto.

1: **while** $|\Sigma| > 1$ **do**
2: Etsi Σ:n kaksi pienimmän todennäköisyyden merkkiä a_1 ja a_2 ja muodosta solmu $a_{1,2}$.
3: $p(a_{1,2}) := p(a_1) + p(a_2)$.
4: $\text{left}(a_{1,2}) := a_1$.
5: $\text{right}(a_{1,2}) := a_2$.
6: $\Sigma := (\Sigma - \{a_1, a_2\}) \cup \{a_{1,2}\}$.
7: **end while**
8: Σ:n viimeinen merkki on koodipuun juuri.



Kuva 3.3: Huffman-koodin muodostaminen.

0.15, 0.2, 0.25 ja 0.3. Merkin A koodisanaksi saadaan 010 ja merkin B koodisanaksi 011. Voidaan todeta, että algoritmin tuottamalla koodilla on teoreeman 3.6.1 mukaiset ominaisuudet.

Huffman-koodaus on käytössä monissa tunnetuissa tiivistysohjelmissa; usein se on yhdistetty johonkin toiseen tiivistysmenetelmään. GNU-projektin tiivistysohjelmassa bzip2 on Huffman yhdistetty Burrows-Wheeler-muunnokseen [4]. Muissa yhteyksissä on yhdistetty myös LZ77 ja Huffman. Huffman-koodausta käytetään nykyisinkin tehokkaamman aritmeettisen koodauksen tilalla, koska siihen ei liity samoja patenttiongelmia kuin aritmeettiseen koodaukseen. Toisaalta algoritmi on huomattavasti helpompi toteuttaa kuin aritmeettinen koodaus ja sen tuottamaa staattista koodia on helpompi käsitellä.

3.7 LZSS-koodaus

Lempel ja Ziv ovat kehittäneet useita tiivistysalgoritmeja. Näistä tunnetuimpia ovat *sanakirjamenetelmät* (dictionary method) LZ77 [44] ja LZ78 [45], joista parannetut versiot ovat LZSS [35] ja LZW [41]. Sanakirjamenetelmät käyttävät sanakirjaa, josta etsitään syötteessä esiintyviä sanoja. Syötteen sanat korvataan viittauksella sanakirjaan. LZ77:ssa ja LZ78:ssa yksi koodaajan tulostama alkio voi siis vastata useampaa syötteen merkkiä. Molemmat merkkijonon sovittamiseen perustuvat menetelmät ovat varsin suosittuja ja toimivia; niiden esittelyn jälkeen merkkijononsovituksesta on tullut tärkeä tiedontiivistyksen osa-alue.

LZ77-menetelmässä käytetään sanakirjana syötteestä jo nähtyä ikkunaa. Ikkunasta etsitään mahdollisimman pitkää sellaista merkkijonoa, joka on identtinen seuraavaksi koodattavan syötteen kanssa. Ikkuna liukuu eteenpäin koodauksen edetessä; LZ77-menetelmää sanotaankin *liukuvan sanakirjan menetelmäksi* (sliding dictionary method). Menetelmässä etsitään sellaista sovituksen paikkaa k, että

$$a_{p-k+(0\ldots l)} = a_{p+(0\ldots l)}, \tag{3.30}$$

missä l pyritään saamaan mahdollisimman suureksi, p on koodaajan nykyinen paikka ja l sovituksen pituus. Yleensä k rajoitetaan laskennallisen tehokkuuden, muistivaatimusten ja tiivistystehon[2] kannalta sopivalle välille. Jos riittävän pitkä sovitus löytyy, koodataan osoitin sovituksen alkuun k ja sovituksen pituus l ja edetään so-

[2] Mitä suurempi vaihteluväli, sen enemmän tilaa tarvitaan sen esittämiseksi.

vituksen pituuden verran. Muussa tapauksessa koodataan osoitin rajatun alueen ulkopuolelle (tai joku sovittu osoittimen arvo, esim. 0) ja seuraava syötteen merkki a_{p+1} sellaisenaan ja siirrytään yksi merkki eteenpäin. [44]

Alkuperäisessä LZ77:ssa [44] koodattiin aina osoitin ja yksi sovituksen jälkeinen merkki; osoitin ja sovituksen pituus koodattiin lähdeaakkoston merkeillä α-kantaisena (radix-α) esityksenä, missä α on lähdeaakkoston merkkien lukumäärä. Storer ja Szymanski [35] tehostivat menetelmää siten, että sovitus ja yksittäinen merkki erotetaan väärän osoittimen sijasta yhdellä bitillä.[3] Osoitin ja pituus kirjoitetaan vain, jos menettelyllä säästetään tilaa verrattuna merkin koodaamiseen sellaisenaan. Pienin sovituksen pituus riippuu koodaajan parametreista. LZSS on esitetty Algorimissa 4.

Esimerkki 3.7.1 Koodataan merkkijono ABCABDCAB LZSS-algoritmilla. Algoritmin toimintaa on havainnollistettu taulukossa 3.5. Kohdatessaan neljännen merkin koodaaja löytää jo nähdystä datasta kahden merkin mittaisen sovituksen kolmen merkin etäisyydeltä ja seitsemännen merkin kohdalla löydetään kolmen merkin mittainen sovitus neljän merkin etäisyydeltä. Koodauksen tulos on ABC(-3,2)D(-4,3).

Syöte	Tulos
A	A
B	B
C	C
A	$(-3, 2)$
B	
D	D
C	$(-4, 3)$
A	
B	

Taulukko 3.5: Esimerkki LZSS-koodauksesta.

Purettaessa on hyödynnettävä aiemmin purettua syötettä. Luetaan tiedostosta seuraava koodi. Jos kyseessä on sovitus, luetaan sovitus jo puretusta syötteestä, muutoin puretaan merkki, joka oli koodattu. Purkaminen onnistuu, koska kaikki data, johon viitataan, on purettu ennen viittausta.

Algoritmi olettaa, että saman merkkijonon toistuminen syötteessä lyhyellä välillä on todennäköistä. Jos näin on, voi se saavuttaa suuriakin tiivistyssuhteita. LZSS:n

[3] Mistä nimi LZSS.

maksimitiivistys riippuu käytetyistä parametreistä eli siitä, kuinka pitkä sovitus voidaan tehdä. Tällainen raja ei ole välttämätön; voidaanhan sovituksen pituus koodata jollain universaalikoodilla. Yleensä sovituksen pituuden tallettamiseen varataan melko pieni määrä bittejä, koska on yleensä järkevää olettaa, että suurin osa sovituksista on lyhyitä. Huffman-koodissa koodattava merkki voidaan pienimmillään tallettaa yhteen bittiin; periaatteessa LZSS voi päästä huomattavasti parempaan tiivistykseen. LZ77 ja sen variantit ovat käytössä tunnetuissa tiivistysohjelmissa (esimerkiksi zip).

Algoritmi 4 LZSS-koodaus

Alkuehto: Merkkijono $a = a_1 a_2 \dots a_n$ on syöte, I ikkunan pituus ja M sovituksen maksimipituus. Yksinkertaisuuden vuoksi oletetaan, että a_i, kun $i \leq 0$, ovat jotain sopivia arvoja (esim. 0).

1: $j := 1$.
2: **while** $j \leq n$ **do**
3: Etsi pisin sellainen merkkijono $\omega = a_{p-I} \dots a_{p+k-I}$, $p = 0, \dots, (I-1)$, $k = 0, \dots, M$, että $\omega \sqsubset a_j \dots a_N$.
4: **if** $\omega = \epsilon$ **then**
5: Koodaa merkki a_j, $j := j + 1$.
6: **else**
7: Koodaa $(p, |\omega|)$, $j := j + |\omega|$.
8: **end if**
9: **end while**

3.8 Ziv-Lempel-koodaus

Kenties tunnetuin Lempelin ja Zivin kehittämistä tiivistysmenetelmistä on LZ78, jonka tunnetuin variantti on Welchin kehittämä LZW [41]. Kun LZ77:ssä käytettiin sanakirjana aiemmin nähtyä dataa, rakennetaan LZW:ssä sanakirjaa jatkuvasti koodauksen aikana. Menetelmää kutsutaan *dynaamisen sanakirjan menetelmäksi* (dynamic dictionary method). LZW on käytössä mm. tunnetussa GIF-tiedostoformaatissa, mutta sen käyttöä rajoittaa Unisysin patentti.

LZW on luonteeltaan mukautuva ja se pystyy yhdellä merkillä esittämään pitkiä sarjoja syötteestä. Se ei tarvitse tietoa lähteen todennäköisyysjakaumasta ennen koodausta, vaan pystyy dynaamisesti sopeutumaan siihen koodauksen aikana. Sopivalle lähteelle LZW pystyy saavuttamaan informaatioteoreettisen alarajan; käytännössä joudutaan kuitenkin tekemään kompromisseja, joissa optimaalisuudesta joudutaan tinkimään. Storer [34] käsittelee käytännön toteutuksen ongelmia ja ratkaisuja.

LZW-algoritmi (Algoritmi 5) perustuu syötteen rekursiiviseen jäsentämiseen erillisiin lohkoihin; samalla nähdyistä lohkoista muodostetaan sanakirja. Lohkoksi valitaan aina pisin sanakirjassa esiintyvä sana, joka esiintyy syötteessä alkaen paikasta, jossa koodaaja on menossa. Aluksi sanakirja on täytetty kaikilla lähdeaakkoston merkeillä ts. jokainen sana sanakirjassa on yhden merkin mittainen. Etsitään sanakirjasta mahdollisimman pitkä sana ω, joka esiintyy tulevan syötteen etuliitteenä ja koodataan sanan indeksi. Edetään syötteessä $|\omega|$ merkkiä. Lisätään sanakirjaan sana ωa, joka koostuu sanakirjasta löydetystä sanasta ja merkistä, johon päädyttiin syötteessä etenemällä. Sanat indeksoidaan samassa järjestyksessä kuin ne on lisätty. [41]

Algoritmi 5 LZW-koodaus

Alkuehto: Σ on lähdeaakkosto, $a = a_1 a_2 \dots a_n$ on syöte.

1: **for all** $x_i \in \Sigma$ **do**
2: Lisää sanakirjaan sana x_i indeksiin i.
3: **end for**
4: $i := |\Sigma|$.
5: $j := 1$.
6: **while** $j \leq n$ **do**
7: Etsi pisin sellainen sanakirjaan kuuluva sana ω, että $\omega \sqsubset a_j \dots a_n$.
8: $j := j + |\omega|$.
9: Tulosta ω:n indeksi sanakirjassa.
10: Lisää sanakirjaan sana ωa_j indeksiin i.
11: $i := i + 1$.
12: **end while**

Esimerkki 3.8.1 Taulukossa 3.6 on esitetty esimerkki LZW:n toiminnasta. Jäsennyksessä merkintä $\omega?$ tarkoittaa, että jäsentäjä on jäsentämässä sanaa ω ja merkintä $\omega a, a?$ sitä, että on jäsennetty ja lisätty sanakirjaan sana ωa ja tulostettu sanan ω indeksi. Aluksi sanakirja on $\{0, 1\}$. Toisen merkin kohdalla ensimmäinen tuntematon merkki on 1, koska sanaa 01 ei löydy sanakirjasta. Pisin sovittunut sana oli 0, joten koodataan sen indeksi ja lisätään 01 sanakirjaan.

Koodaajalla on purkamisen kannalta tärkeä *viimeinen-ensimmäinen-ominaisuus* (last-first property), joka tarkoittaa sitä, että viimeksi lisätyn sanan viimeinen merkki on seuraavan jäsennetyn sanan ensimmäinen merkki. Ominaisuutta tarvitaan tilanteessa, jossa purkaja saa syötteekseen indeksin, jota ei ole vielä sanakirjassa. [41]

Syöte	Tulos	Sanakirja	Jäsennys
0		0, 1	0?
1	0	0, 1, 01	01,1?
1	1	0, 1, 01, 11	11,1?
0	1	0, 1, 01, 11, 10	10,0?
0	0	0, 1, 01, 11, 10, 00	00,0?
1		0, 1, 01, 11, 10, 00	01?
1	2	0, 1, 01, 11, 10, 00, 011	011, 1?
0		0, 1, 01, 11, 10, 00, 011	10?
0	4	0, 1, 01, 11, 10, 00, 011, 100	100, 0?
1		0, 1, 01, 11, 10, 00, 011, 100	01?
EOF	2	0, 1, 01, 11, 10, 00, 011, 100	01

Taulukko 3.6: Esimerkki LZW-koodauksesta

Purkaja lukee indeksin ja tulostaa sen sisältämän merkkijonon ω. Purkamista jatketaan lukemalla seuraava indeksi ja lisäämällä sanakirjaan sana ωa, joka muodostetaan edellisestä sanasta ω ja seuraavan indeksin sisältämän sanan ensimmäisestä merkistä a. Purkaja voi saada syötteekseen indeksin, joka ei ole sanakirjan sisällä. Tämä johtuu siitä, että koodaaja on jo lisännyt sanan sanakirjaan, mutta purkaja ei ole voinut sitä tehdä, koska ei vielä tiedä sanan viimeistä kirjainta. Ongelma ilmenee, kun koodaaja on saanut syötteekseen $x\omega x\omega x$, missä x on yksittäinen merkki ja ω joko tyhjä tai sellainen merkkijono, että $x\omega$ on jo sanakirjassa. Tilanteen ratkaisemiseksi on sovellettava viimeinen-ensimmäinen-ominaisuutta, jolloin edellisen puretun merkkijonon ensimmäinen kirjain on purettavan merkkijonon viimeinen, näin ongelma on ratkaistu. [41]

Esimerkki 3.8.2 Purettaessa esiintyvää ongelmaa ja sen ratkaisua on havainnollistettu taulukossa 3.7; viimeinen-ensimmäinen-ominaisuuden käyttö on merkitty (LF):llä. Kun purkaja saa syötteeksi indeksin 2, jolla ei ole vielä sanaa sanakirjassa käytetään viimeinen-ensimmäinen-ominaisuutta ja otetaan sen viimeiseksi kirjaimeksi edellisella kierroksella puretun sanan (indeksi 0) ensimmäinen merkki 0.

Algoritmin toteutuksessa ei ole järkevää tallettaa sanakirjaan jokaista merkkijonoa sellaisenaan, koska ne veisivät liikaa tilaa ja merkkijonojen etsimisestä tulisi vaikeaa. Merkkijonot voidaan tallettaa päällekkäin puuhun siten, että jokainen polku juuresta johonkin solmuun sisältää yhden merkkijonon ja jokainen lapsisolmu

Tiivistys			
Syöte	Tulos	Sanakirja	Jäsennys
0		0, 1	0?
0	0	0, 1, 00	00, 0?
0		0, 1, 00	00?
0	2	0, 1, 00, 000	000, 0?
0		0, 1, 00, 000	00?
0		0, 1, 00, 000	000?
EOF	3	0, 1, 00, 000	000
Purkaminen			
Syöte	Tulos	Sanakirja	Jäsennys
0	0	0, 1, 0?	0, 0?
2	00	0, 1, 00(LF), 00?	00, 0?
3	000	0, 1, 00, 000(LF), 000?	000, 0?

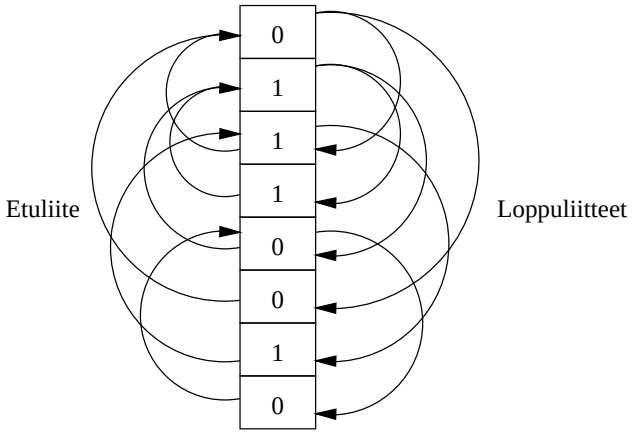
Taulukko 3.7: Viimeinen-ensimmäinen-ominaisuuden käyttö.

merkkijonon seuraavan merkin.[4] Järjestelyllä on helpompaa etsiä pisin sovitus, koska oikean polun seuraaminen karsii automaattisesti pois merkkijonot, jotka eivät sovitu syötteeseen. Tämän tietorakenteen avulla saadaan algoritmin aikavaatimus lineaariseksi syötteen pituuden suhteen, olettaen, että oikean lapsisolmun valinta voidaan tehdä vakioajassa. Kuvassa 3.4 on esitetty toteutuksen tietorakenteen tila taulukossa 3.6 esitetyn esimerkin lopussa. Tehtäväksi jää vielä algoritmin tuottamien indeksien koodaaminen; tyypillisesti käytetään pienintä määrää bittejä, jolla voidaan esittää sanakirjan koko. Koodaukseen voidaan hyvin käyttää myös jotain universaalikoodia.

3.9 Aritmeettinen koodaus

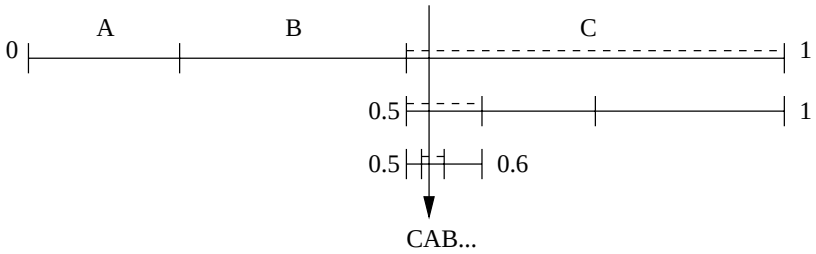
Gershon ja Grayn [11] mukaan Eliasin julkaisemattomaan koodausmenetelmään pohjautuva aritmeettinen koodaus on teoreettisesti optimaalinen; se saavuttaa aina informaatioteoreettisen alarajan koodisanan pituudelle. Menetelmää ovat kehittäneet mm. Rissanen ja Langdon [31], Witten *et al.* [43], Fenwick [10] ja Moffat [27]. Ensimmäisen monimerkkisellä aakkostolla toimivan aritmeettisen koodaajan käytännön toteutuksen esittivät Witten *et al.* [43] vuonna 1987, jonka jälkeen aritmeettisesta koodauksesta tuli suosittu entropian koodauksen menetelmä. Fenwick

[4] Tällaisen tietorakenteen nimi on *trie* [34].



Kuva 3.4: LZW toteutuksen tietorakenne.

[10] ratkoo mukautuvaan aritmeettiseen koodaukseen liittyviä ongelmia ja Moffat [27] esittää parannuksia koodaajaan ja yhteenvedon menetelmän kehityksestä.



Kuva 3.5: Puhdas aritmeettinen koodaus.

Aritmeettisen koodauksen ideana on lähettää viesti reaalilukuna x väliltä $[0, 1)$. Väli jaetaan merkkien todennäköisyyden perusteella osaväleihin niin, että kunkin osavälin pituus kuvaa merkin todennäköisyyttä. Merkin koodaamiseksi tarvitsee lähettää mikä tahansa luku, joka kuuluu merkkiä kuvaavalle välille. Koko viestin koodaamiseksi välit jaetaan aina uudelleen rekursiivisesti ja lähetetään tarkemmin rajoitettu luku. Jos koodaaja on menossa välillä $I = [a, b)$ ja koodattavaa lukua vastaa välin $[0, 1)$ jaossa väli $[c, d)$, niin luvun on kuuluttava välille

$$I' = \left[a + \frac{b-a}{c}, a + \frac{b-a}{d} \right). \tag{3.31}$$

Purettaessa, jos purkaja on menossa välillä $I = [a, b)$ ja se on lukenut luvun $x \in$

$[a, b)$, niin luku alkuperäisellä osavälijaolla on

$$x' = \frac{x - a}{b - a}. \tag{3.32}$$

Nyt purkajan on vain etsittävä se alkuperäisen jaon väli, jolle x' kuuluu ja lähdeaakkoston merkki, jota kyseinen väli vastaa.

Esimerkki 3.9.1 Koodausta on havainnollistettu kuvassa 3.5, jossa on aakkostona $\{A, B, C\}$ ja merkkien todennäköisyyksinä $p(A) = 0.2$, $p(B) = 0.3$ ja $p(C) = 0.5$. Välit on jaettu siten, että merkkiä A vastaa väli $[0, 0.2)$, merkkiä B väli $[0.2, 0.5)$ ja merkkiä C väli $[0.5, 1.0)$. Koodattava viesti alkaa merkeillä CAB. Ensimmäinen merkki C on koodattava välille $[0.5, 1)$, joka jaetaan osiin. Merkkejä vastaaviksi osaväleiksi saadaan $[0.5, 0.6)$, $[0.6, 0.75)$ ja $[0.7, 1.0)$. Seuraava merkki on A, jolloin on koodattava väli $[0.5, 0.6)$. Jatkamalla menettelyä huomataan, että viestin alkuosan CAB esittämiseksi voidaan lähettää mikä tahansa luku väliltä $[0.52, 0.55)$.

Tärkein aritmeettisen koodaajan etu on sen joustavuus: se erottaa lähteen tilastollisen mallin koodaajasta täysin. Aina kun seuraavaa koodattavaa väliä jaetaan, voidaan vaihtaa jakoa koodaajasta riippumatta. Etu on merkittävä, koska esimerkiksi Huffman-koodaajasta mallin erottaminen on lähes mahdotonta. Aritmeettisen koodaajan optimaalisuus ei ole niin merkittävää kuin voisi helposti luulla; Huffman-koodikin on useissa tapauksissa lähes optimaalista. Suurin ero Huffman-koodiin tulee tilanteessa, jossa jonkin merkin todennäköisyys on lähellä ykköstä. Tässä tapauksessa Huffman-koodi on huonoimmillaan, koska sen on esitettävä jokainen lähdeaakkoston merkki vähintään yhdellä bitillä. Aritmeettisessa koodauksessa ei tätä rajoitusta ole. Käytännössä suuriin tiivistyssuhteisiin päästään yleensä vain laadukkaiden tilastollisten mallien avulla. [16]

Suurin aritmeettisen koodaajan ongelma on sen hitaus, täyden tarkkuuden aritmeettinen koodaaja on hitaampi kuin Huffman- tai Ziv-Lempel-koodaus. Toinen koodaajan ongelma on se, ettei sitä voida toteuttaa rinnakkaislaskennan avulla, koska seuraava koodisana riippuu edellisen koodatun merkin vaikutuksesta koodaajan tilaan. Koodista ei myöskään selviä tiedoston loppuminen; tämä ongelma ratkaistaan yleensä erillisen EOF-merkin lisäyksellä lähteen aakkostoon. [16]

Aritmeettisen koodaajan toteuttaminen sellaisenaan ei käytännössä onnistu, koska se tarvitsisi äärettömän tarkkaa laskentatarkkuutta. Tarkkuuden tarve kasvaa koko ajan koodattavan viestin kasvaessa. Witten *et al.* [43] esittivät yksinkertaisen ratkaisun ongelmaan. Heidän menetelmässään koodaajan osaväliä laajennetaan

sopivasti ettei laskutarkkuus pääse kasvamaan liian suureksi. Koodaajan tilan tallentamiseksi tarvitaan vain 3 lukua (ylä- ja alaraja ja laskuri). Jos koodattava väli sisältyy väliin $[0, 1/2)$, niin seuraava bitti koodattavan luvun binääriesityksessä on 0, vastaavasti jos väli sisältyy väliin $[1/2, 1)$, niin seuraava bitti koodattavan luvun binääriesityksessä on 1. Jos väli sisältyy väliin $[1/4, 3/4)$, niin seuraavan koodattavan bitin jälkeinen bitti on arvoltaan päinvastainen. Koodattavaa väliä voidaan tämän luokituksen jälkeen laajentaa kaksinkertaiseksi laajentamalla lineaarisesti suurempi väli, jolle se kuului väliksi $[0, 1)$. Viimeisen vaihtoehdon tapahtumakerroista on lisäksi pidettävä kirjaa laskurilla. Kun tiedetään seuraava bitti, pitää koodata laskurin verran bittejä, joiden arvo on päinvastainen ja nollata laskuri.

Käytännön toteutuksessa reaalilukujen tilalla käytetään *kiinteäpilkkulukuja* (fixed point number) eli lukuväli $[0, 1)$ skaalataan väliksi $[0, 2^b)$ ja käytetään esitykseen vain välille kuuluvia kokonaislukuja, jolloin lukua 1 voisi vastata esimerkiksi $2^{16} = 65536$, tällöin lukua $1/2$ vastaisi $2^{16}/2 = 32768$. Mitä suurempaa määrää bittejä käytetään, sitä tarkemmaksi koodaaja tulee ja tarkkuuden kasvaessa kasvaa myös tiivistysteho, koska lähteen merkkien todennäköisyydet voidaan esittää tarkemmin.

Moffat [27] esittää parannetun version koodaajasta (Algoritmit 6, 7 ja 8), jossa koodaajan väli esitetään sen alarajan L ja pituuden R avulla, jolloin saadaan säästöjä aritmeettisten operaatioiden määrässä. Algoritmit on esitetty samassa muodossa; operaatioiden järjestyksellä on merkitystä, koska kaikki tulokset pyöristetään kokonaisluvuiksi. Käytännön toteutuksessa todennäköisyyksiä käsitellään kumulatiivisen frekvenssitaulukon[5] avulla. Koodaajan tilaksi alussa asetetaan $L := 0$ ja $R := 2^{b-1}$ ja koodauksen lopussa kirjoitetaan L:n b bittiä siten, että ensimmäisen bitin jälkeen kirjoitetaan c kappaletta bittejä, joiden arvo on päinvastainen ensimmäisen bitin kanssa, ja loput bitit sellaisenaan. Purkajan tilaksi alustetaan $R := 2^{b-1}$ ja D:n arvoksi luetaan b ensimmäistä bittiä tiedostosta. Purkaminen tapahtuu kutsumalla ensin kohdefrekvenssin purkua (Algoritmi 7) ja etsimällä sitten koodattu väli, jolle kohdefrekvenssi kuuluu. Purkaja päivitetään koodatulla välillä ja uudella frekvenssitaulukon ylärajalla, jos se on muuttunut (Algoritmi 8).

Käytännön toteutuksen aiheuttamat menetykset optimaaliseen ratkaisuun verrattuna on osoitettu mitättömiksi. Jos reaalilukuväli $[0, 1)$ skaalataan kokonaislukuväliksi $[0, N)$, niin ylimääräinen koodin pituus syötemerkkiä kohti on pienempi kuin $4/(N \ln 2)$ bittiä. Tyypillisellä N:n arvolla 65536 ylimääräinen koodin pituus on vähemmän kuin 10^{-4} bittiä merkkiä kohti. Kun EOF-merkki lisätään aakkostoon,

[5] Taulukossa on merkkien esiintymien kertymäfunktio; funktio voi olla joko todellinen tai arvioitu.

Algoritmi 6 Aritmeettinen koodaus

Alkuehto: $I = [l, h)$ on koodattava väli kumulatiivisessa frekvenssitaulukossa ja t on yhteenlaskettu merkkien määrä (todennäköisyytenä väli on $[l/t, h/t)$). Koodaaja on menossa välillä, jonka alaraja on L ja pituus R, ja c on bittilaskuri. Koodaukseen käytetään b bittiä.

1: $r := R/t$.
2: $L := L + rl$.
3: **if** $h < t$ **then**
4: $\quad R := r(h - l)$.
5: **else**
6: $\quad R := R - rl$.
7: **end if**
8: **while** $R \leq 2^{b-1}$ **do**
9: $\quad$ **if** $L + R \leq 2^{b-1}$ **then**
10: $\quad\quad$ Tulosta 0-bitti ja c kpl 1-bittejä.
11: $\quad\quad c := 0$.
12: $\quad$ **else if** $L \geq 2^{b-1}$ **then**
13: $\quad\quad L := L - 2^{b-1}$.
14: $\quad\quad$ Tulosta 1-bitti ja c kpl 0-bittejä.
15: $\quad\quad c := 0$.
16: $\quad$ **else**
17: $\quad\quad L := L - 2^{b-2}$.
18: $\quad\quad c := c + 1$.
19: $\quad$ **end if**
20: $\quad L := 2L$.
21: $\quad R := 2R$.
22: **end while**

Algoritmi 7 Kohdefrekvenssin purkaminen

Alkuehto: Purkaja on menossa välillä, joka pituus on R. D on koodatun luvun etäisyys välin alarajasta ja t on frekvenssitaulukon yläraja.

Jättöehto: Palautettu arvo on kohdefrekvenssi, jonka avulla voidaan selvittää koodattu väli frekvenssitaulukosta.

1: $r := R/t$;
2: **return** $\min\{t - 1, D/r\}$.

Algoritmi 8 Aritmeettinen dekoodaus

Alkuehto: $I = [l, h)$ on edellisellä kierroksella koodattu väli kumulatiivisessa frekvenssitaulukossa ja t on yhteenlaskettu merkkien määrä. Purkaja on menossa välillä, jonka leveys on R. Koodaukseen käytetään b bittiä. D on puretun arvon etäisyys välin alareunasta. Luku r on asetettu edellisessä kutsussa kohdefrekvenssin purkuun.

1: $D := D - rl$.
2: **if** $h < t$ **then**
3: $\quad R := r(h - l)$.
4: **else**
5: $\quad R := R - rl$.
6: **end if**
7: **while** $R \leq 2^{b-2}$ **do**
8: $\quad R := 2R$.
9: $\quad D := 2D +$ lue bitti.
10: **end while**

lisääntyy t merkin mittaisen syötteen koodin pituus vähemmän kuin $8t/(N \ln 2) + \lg N + 7$ bittiä. [16]

Aritmeettisesta koodauksesta saadaan helposti mukautuva, koska koodaaja ja purkaja tarvitsevat vain koodattavia lukuvälejä. Ongelmaksi muodostuu siis kumulatiivisen frekvenssitaulukon tai jonkin muun vastaavan rakenteen ylläpitäminen. Fenwick [10] esittää ongelmaan elegantin ratkaisun, jossa merkin todennäköisyyden päivittäminen, merkin välin etsiminen ja kohdefrekvenssin välin etsiminen voidaan suorittaa ajassa $O(\log n)$ frekvenssitaulukon merkkien lukumäärän suhteen.

4 Signaalien tiivistäminen

Signaalien tiivistäminen on paljon tutkittu digitaalisen signaalinkäsittelyn alue. Viime aikoina huomiota on saanut varsinkin musiikin tiivistäminen. Signaalinkäsittely on tietojenkäsittelytieteen osa-alue, jonka tavoitteena on muokata signaaleita sovelluksesta riippuvalla tavalla. Usein on kyse signaalin vahvistamisesta, kohinan poistamisesta tai signaalin analysoinnista. Yleisesityksiä signaalinkäsittelystä ovat kirjoittaneet mm. Defatta *et al.* [6], Lyons [26] ja Proakis ja Manolakis [30]. Kraniauskas [24] käsittelee signaalinkäsittelyn matemaattista teoriaa sekä jatkuvien että diskreettien signaaleiden tapauksessa. Taylor [37] keskittyy suodatinten suunnitteluun ja antaa hyvän katsauksen lineaarisista muunnoksista. Orfanidis [28] käsittelee satunnaisten signaalien käsittelyä. Alexander [1] ja Haykin [13] käsittelevät mukautuvia suodattimia ja niiden sovelluksia ja signaalinkäsittelyä epästationäärisessä ympäristössä. Gersho ja Gray [11] käsittelevät signaalien tiivistämistä.

Tunnettuja signaalien tiivistämisen menetelmiä ovat skalaarikvantisointi, differentiaalinen pulssikoodimodulaatio ja muunnoskoodaus. Viimeaikoina ovat huomiota paljon saaneet aallokkeisiin perustuvat muunnoskoodauksen menetelmät, joita tässäkin käsitellään.

4.1 Signaalinkäsittelyn teoriaa

Proakis ja Manolakis [30] määrittelevät *signaalin* fyysiseksi arvoksi, joka muuttuu ajan tai jonkin muun muuttujan mukana. Matemaattisesti signaali voidaan esittää yhden tai useamman muuttujan funktiona. Esimerkiksi $x(t) = 2t$ on signaali, joka muuttuu lineaarisesti ajan funktiona. Signaalit voivat olla *moniulotteisia*; esimerkiksi kuvat voidaan tulkita kaksiulotteisiksi signaaleiksi. Signaalilla voi myös olla useita *kanavia*, jolloin se saa vektoriarvoja:

$$\mathbf{x}(t) = (x_1(t), x_2(t), \ldots, x_n(t))^T. \tag{4.1}$$

Jatkuvat signaalit ovat jatkuvien muuttujien funktioita ja diskreetit signaalit diskreettien muuttujien funktioita. Jatkuvia signaaleita ei voida käsitellä digitaalisesti, joten ne on muutettava diskreettiin muotoon. Ensin jatkuva signaali *näytteistetään* (sampling), näytteistetty signaali *kvantisoidaan* (quantization) ja lopuksi kvantisoinnin tulos koodataan. Kun puhutaan fyysisistä signaaleista, koko prosessia kutsutaan A/D-muunnokseksi (analogia/digitaali) ja käytettyä laitteistoa A/D-muuntimeksi. [30]

Diskreetti signaali saadaan jatkuvasta signaalista näytteistämällä:

$$x(n) = x(nT),\ n \in \mathbb{Z}, \tag{4.2}$$

missä T on näytteenottovälin pituus; tällöin *näytteenottotaajuus* f_s on $1/T$. Diskreetti signaali $x(n)$ on määritelty vain n:n kokonaislukuarvoilla. Näytteitä on otettava riittävän tihein välein, jotta signaalin kaikki vaihtelut saadaan talteen. Riittävä välien tiheys saadaan nk. *näytteistysteoreemasta* (sampling theorem). Sen mukaan näytteenottotaajuuden pitää olla vähintään kaksi kertaa kertaa niin suuri, kuin suuritaajuisimman näytteistettävässä signaalissa esiintyvän komponentin taajuus. Niinpä järjestelmä, jonka näytteenottotaajuus on f_s, voi esittää tarkasti vain signaaleita, joiden komponenttien taajuus on alle $f_s/2$. Taajuutta $f_s/2$ sanotaan *Nyquistin taajuudeksi*. Näytteistettävästä signaalista pitäisi ennen näytteistämistä suodattaa pois taajuudet, jotka ylittävät Nyquistin taajuuden, tai tapahtuu *laskostumista* (aliasing). Laskostuminen tarkoittaa korkeampitaajuisen komponentin ilmenemistä matalampitaajuisena komponenttina näytteistetyssä signaalissa. Jotta tarkasteltava informaatio säilyisi mahdollisimman laadukkaana, tämä ei tietenkään ole toivottavaa. [30]

Digitaaliseen muotoon signaali saadaan kvantisoimalla:

$$y(n) = Q[x(n)], \tag{4.3}$$

missä Q on kvantisointioperaatio. Kvantisoinnin jälkeen signaali voi yleensä[1] saada äärellisen määrän arvoja eli kvantisoinnissa muutetaan signaalin arvoalue diskreetiksi [30]. Kvantisointia käsitellään tarkemmin myöhemmin signaalien tiivistyksen yhteydessä.

Tyypillisimmät signaalille tehtävät operaatiot ovat skalaarilla kertominen (4.4), yhteenlasku (4.5), kertolasku (4.6) ja siirto ajassa (4.7):

$$y(n) = ax(n), \tag{4.4}$$
$$y(n) = x_1(n) + x_2(n), \tag{4.5}$$
$$y(n) = x_1(n)x_2(n), \text{ ja} \tag{4.6}$$
$$y(n) = z^k x(n) = x(n+k),\ k \in \mathbb{Z}. \tag{4.7}$$

[1] Teorian tutkimisen kannalta on ollut hyödyllistä tutkia kvantisointia, jonka arvoalue on ääretön.

Usein tavattavia operaatioita ovat myös näytteenottotaajuuden muuttaminen ylös- (4.8) tai alaspäin (4.9). (upsampling, downsampling) [30]:

$$y(m) = x(n) \uparrow k, m = n/k, \text{ ja} \tag{4.8}$$

$$y(m) = x(n) \downarrow k, m = kn. \tag{4.9}$$

Signaalit voidaan esittää muiden signaalien avulla. Tyypillisimpiä näistä ovat *impulssi* (impulse), askel (unit step) ja kompleksit eksponentiaalifunktiot:

$$\delta(n) = \begin{cases} 1, \ n = 0 \\ 0, \ n \neq 0 \end{cases}, \tag{4.10}$$

$$u(n) = \begin{cases} 1, n \geq 0 \\ 0, n < 0 \end{cases}, \text{ ja} \tag{4.11}$$

$$x(n) = re^{i\theta n}, \tag{4.12}$$

joista viimeisiä käytetään Fourier-analyysissä. Esimerkiksi signaali $x(n)$ voidaan esittää yksikköimpulssien avulla seuraavasti:

$$x(n) = \sum_{i=-\infty}^{\infty} \delta(n-i)x(i). \tag{4.13}$$

Tällöin signaalin sanotaan olevan esitettynä *aika-alueella* (joskus puhutaan myös ns. standardikannasta). [30]

Määritelmä 4.1.1 Diskreetin epäjaksollisen signaalin $x(n)$ Fourier-muunnos $X(\omega)$ on

$$X(\omega) = \mathfrak{F}\{x(n)\} = \sum_{n=-\infty}^{\infty} x(n)e^{-i\omega n}. \tag{4.14}$$

Fourier-muunnos on määritelty signaalille, jos sen itseisarvojen summa on äärellinen eli $\sum_{-\infty}^{\infty} |x(n)| < \infty$. Diskreetin signaalin Fourier-muunnos on 2π-jaksollinen, ts. $X(\omega) = X(\omega + 2\pi k)$. Fourier-muunnos esittää signaalin eritaajuisten ja -vaiheisten siniaaltokomponenttien lineaarikombinaationa. $X(\omega)$ esittää signaalin *taajuusalueella*. Jaksollisten signaalien tapausta käsitellään myöhemmin diskreetin Fourier-muunnoksen yhteydessä. Fourier-muunnosta käytetään mm. LTI-suodattimien omi-

naisuuksien tutkimiseen. [30]

Määritelmä 4.1.2 *Suodatin* (filter) on järjestelmä $\mathcal{T}$, joka tuottaa toisen signaalin syötteenään saamasta signaalista:

$$y(t) = \mathcal{T}\left[x(t)\right]. \tag{4.15}$$

Määritelmä 4.1.3 Järjestelmä $\mathcal{T}$ on *lineaarinen ja ajasta riippumaton* (linear time invariant, LTI), jos ja vain jos

$$\mathcal{T}\left[a_1 x_1(n) + a_2 x_2(n)\right] = a_1 \mathcal{T}\left[x_1(n)\right] + a_2 \mathcal{T}\left[x_2(n)\right], \text{ ja} \tag{4.16}$$

$$\mathcal{T}\left[z^k x(n)\right] = z^k \mathcal{T}\left[x(n)\right]. \tag{4.17}$$

LTI-järjestelmät ovat tavallisimpia kaikista signaalinkäsittelyn järjestelmistä. Jos järjestelmä $\mathcal{T}$ on LTI, voidaan se täydellisesti kuvata *impulssivasteen* (impulse response) $h(n)$ avulla; tämän ominaisuuden takia LTI-järjestelmien analysointi on helppoa. [30]

Määritelmä 4.1.4 LTI-järjestelmän $\mathcal{T}$ impulssivaste on

$$h(n) = \mathcal{T}\left[\delta(n)\right]. \tag{4.18}$$

Määritelmä 4.1.5 LTI-järjestelmä $\mathcal{T}$ on *stabiili*, eli se tuottaa äärellisen tulossignaalin jokaisella äärellisellä syötesignaalilla, jos ja vain jos

$$\sum_{-\infty}^{\infty} |h(n)| < \infty. \tag{4.19}$$

Määritelmä 4.1.6 LTI-järjestelmää $\mathcal{T}$ sanotaan *kausaaliseksi*, jos

$$h(n) = 0, \text{ kun } n < 0. \tag{4.20}$$

Kausaalinen järjestelmä pystyy tuottamaan tulossignaalin arvon heti, kun tiedetään syötesignaalin arvo. Jos järjestelmä ei ole kausaalinen, pystytään se käytännössä toteuttamaan vain viiveen avulla. [30]

LTI-järjestelmän vaste mielivaltaiseen signaaliin $x(n)$ saadaan helposti selville jakamalla $x(n)$ komponentteihin, jotka koostuvat siirretyistä ja skaalatuista impuls-

seista. Tulossignaali voidaan ilmaista impulssivasteen avulla:

$$\mathcal{T}\left[x(n)\right] = \sum_{i=-\infty}^{\infty} \mathcal{T}\left[x(i)\delta(n-i)\right] = \sum_{i=-\infty}^{\infty} x(i)h(n-i). \tag{4.21}$$

Viimeistä summaa kutsutaan *konvoluutioksi*, ja sitä merkitään yleensä tähdellä. LTI-systeemin tapauksessa voidaan tulossignaali siis esittää muodossa:

$$y(n) = x(n) * h(n). \tag{4.22}$$

Konvoluutio vastaa signaalien Fourier-muunnosten kertomista, jolloin suodattimen taajuusvaste voidaan määrittää sen impulssivasteen Fourier-muunnoksesta. [30]

Tyypillisiä LTI-järjestelmiä ovat *lineaariset suodattimet*. Ne jaetaan kahteen luokkaan impulssivasteen pituuden perusteella. Jos suodattimen impulssivaste on pituudeltaan äärellinen, kutsutaan suodatinta *äärellisen impulssivasteen suodattimeksi* (finite impulse response, FIR), muutoin suodatin on *äärettömän impulssivasteen suodatin* (infinite impulse response, IIR). FIR- ja IIR-suodattimet ilmaistaan usein differenssiyhtälön muodossa.

$$y(n) = \sum_{i=0}^{N} a_i x(n-i) + \sum_{j=1}^{M} b_j y(n-j). \tag{4.23}$$

Jos kertoimet b_j ovat nollia, niin suodatinta kutsutaan *liukuvan keskiarvon suodattimeksi* (moving average, MA), tosin sille on myös muita nimiä[2]. Jos kertoimet a_i ovat nollia, niin suodatinta kutsutaan *autoregressiiviseksi* (AR). Silloin, kun suodattimella on kumpiakin kertoimia, sitä kutsutaan *autoregressiiviseksi liukuvaksi keskiarvoksi (ARMA)*. Nimien sekavuus johtuu siitä, että tilastotieteellä ja signaalinkäsittelyllä on oma terminologiansa samoille asioille. Signaalinkäsittelyssä AR-suodatin on *all-pole-suodatin*, MA on *all-zero-suodatin* ja ARMA:a voidaan kutsua rekursiiviseksi suodattimeksi. Yleensä, jos kertoimet b_j eivät ole nollia, on suodatin äärettömän impulssivasteen suodatin, joskaan tämä ei aina pidä paikkaansa. Toisaalta jos kertoimet b_j ovat nollia, niin suodatin on äärellisen impulssivasteen suodatin, jos kertoimien a_i määrä N on äärellinen. [30]

Lineaarisista suodattimista halutaan yleensä tietää niiden *taajuusvaste*. Taajuusvasteella tarkoitetaan vaikutusta eri taajuuksien sinimuotoisiin signaaleihin (tar-

[2] mm. transversal filter, all-zero -filter, ei-rekursiivinen suodatin

kemmin ottaen kompleksisiin eksponentiaalifunktioihin). LTI-systeemin vaste siniaaltosignaaliin on aina saman taajuuden siniaaltosignaali, jonka vaihe ja amplitudi ovat mahdollisesti muuttuneet. Taajuusvaste siis tarkoittaa systeemin vastetta eri taajuuksisten siniaaltojen vaiheeseen ja amplitudiin. Jos tiedetään, että häiriötä (esim. kohinaa) esiintyy tietyllä kaistalla ja tarvittava informaatio on eri kaistalla, voidaan suunnitella suodatin, joka vaimentaa häiriön. Tyypillisiä suodattimia ovat alipäästösuodatin, joka vaimentaa jonkin taajuuden ylittäviä signaalin komponenttejä, ja ylipäästösuodatin, joka vaimentaa jonkin taajuuden alittavia signaalin komponentteja. Suodattimien suunnitteluun ja ominaisuuksiin ei tässä tarkemmin puututa; suunnittelua on käsitelty kattavasti kirjallisuudessa [6, 13, 24, 26, 28, 30, 37]. Suodatinalgoritmit toimivat yleensä lineaarisessa ajassa syötteen pituuden suhteen.

Signaaleita analysoidaan myös tilastollisesti, jolloin signaali oletetaan *stokastiseksi prosessiksi*. Signaalista tarvitaan yleensä tietää niiden keskiarvo μ_x, varianssi $\sigma_x^2 = E[(x-\mu_x)^2]$ ja toinen momentti $E[x^2]$, missä $E[x]$ tarkoittaa x:n odotusarvoa. Monesti oletetaan, että signaalien keskiarvo on nolla; niin tehdään jatkossa tässäkin. Tilastollisesti signaaleja voidaan verrata toisiinsa korrelaation avulla. [28]

Määritelmä 4.1.7 Signaalien $x(n)$ ja $y(n)$ välinen korrelaatio on

$$r_{xy}(l) = E[x(n)y(n-l)] = E[x(n+l)y(n)],\ l = 0, \pm 1, \pm 2, \ldots \tag{4.24}$$

Korrelaatiolla on voimassa $r_{xy}(l) = r_{yx}(-l)$.

Määritelmä 4.1.8 Signaalin $x(n)$ autokorrelaatiofunktio on

$$r_{xx}(l) = E[x(n)x(n-l)],\ l = 0, \pm 1, \pm 2, \ldots. \tag{4.25}$$

Jos oletetaan että μ_x on nolla, niin $r_{xx}(0) = \sigma_x^2$. Signaalia sanotaan *stationääriseksi*, jos autokorrelaatiofunktio ei muutu ajan myötä. [28]

4.2 Signaalien tiivistämisestä

Signaalien tiivistämiseen voidaan periaatteessa käyttää tunnettuja tiivistämisen menetelmiä. Yleensä ne eivät toimi kovin hyvin, koska mitatut signaalit sisältävät kohinaa, joka tekee vaikeaksi riippuvuuksien löytämisen esimerkiksi merkkijononsovitukseen perustuvilla tekniikoilla.

Häviöllisen tiivistyksen tapauksessa tiivistyksen laatua voidaan objektiivisesti

mitata *virheen itseisarvon keskiarvon* (mean absolute error, MAE), keskineliövirheen (mean squared error, MSE) ja normalisoidun keskineliövirheen neliöjuuren (normalized root mean squared error, NRMSE) avulla. Usein biosignaaleja koskevissa artikkeleissa NRMSE ilmoitetaan prosentteina, jolloin metriikasta käytetään nimeä PRD (percent RMS difference).

Määritelmä 4.2.1 Olkoon $x(n)$ signaali, jonka pituus on N näytettä ja $\hat{x}(n)$ tiivistetystä informaatiosta rekonstruoitu $x(n)$:n approksimaatio, tällöin virhemetriikat MAE, MSE ja NRMSE $x(n)$:n ja $\hat{x}(n)$:n välillä ovat

$$\mathrm{MAE} = \frac{1}{n}\sum_{n=0}^{N} \left| x(n) - \hat{x}(n) \right|, \tag{4.26}$$

$$\mathrm{MSE} = \frac{1}{n}\sum_{n=0}^{N} \left[x(n) - \hat{x}(n) \right]^2 \text{ ja} \tag{4.27}$$

$$\mathrm{NRMSE} = \sqrt{\frac{\sum_{n=0}^{N} \left[x(n) - \hat{x}(n) \right]^2}{\sum_{n=0}^{N} \left[x(n) \right]^2}}. \tag{4.28}$$

Nämä metriikat eivät kuitenkaan pysty kuvaamaan mahdollista informaation häviämistä kuin epäsuorasti. Laadunarviointi on parempi tehdä sovelluksen näkökulmasta mittamaalla eroja sovelluksen suorituskyvyssä tiivistettyjen ja alkuperäisten signaalien välillä. Usein kuitenkin tämä analyysi jätetään tekemättä ja turvaudutaan objektiivisiin metriikoihin.

Ernen [8, 9] mukaan subjektiivista arviointia joudutaan tekemään korkealaatuisessa musiikin tiivistämisessä. Yksinkertaiset metriikat mittaavat vain signaalin muodon säilymistä, eivät muita ominaisuuksia. Laadukkaan tuloksen saavuttamiseksi tiedon karsinta perustuu kokeellisesti mitattuihin psykoakustisiin malleihin. Tällöin tavoitellaan *havainnon laatua* (perceptual quality).

Tiivistettäviä signaaleja mitattaessa on käytettävä sopivaa näytteenottotaajuutta. On saatava talteen tarvittava informaatio, mutta vältettävä turhan informaation keräämistä. Näytteenottoresoluutio on valittava sovelluksen kannalta riittäväksi. Sopivan näytteenottotaajuuden valintaa voidaan pitää periaatteessa häviöttömänä tiivistysmenetelmänä ja sopivan resoluution valintaa häviöllisenä. [11]

Signaalien tiivistyksen menetelmistä käsittelen skalaarikvantisointia, differentiaalista pulssikoodimodulaatiota ja lineaarisiin muunnoksiin perustuvia menetelmiä. Signaalien tiivistyksen menetelmät yhdistetään tunnettuihin tiedon tiivistyksen menetelmiin.

4.3 Häviötön DPCM

Yksi tunnetuimmista signaalin tiivistyksen menetelmistä on *differentiaalinen pulssikoodimodulaatio* (differential pulse code modulation, DPCM). Sitä on sovellettu mm. puheen tiivistykseen. Menetelmän ideana on yrittää ennustaa seuraavaa näytettä edellisistä ja koodata ennustusvirhe. Ennustusvirhesignaalin varianssi pyritään saamaan pienemmäksi kuin alkuperäisen signaalin varianssi. Ennustustaminen vastaa tilastollista mallintamista tiivistysjärjestelmässä. [11]

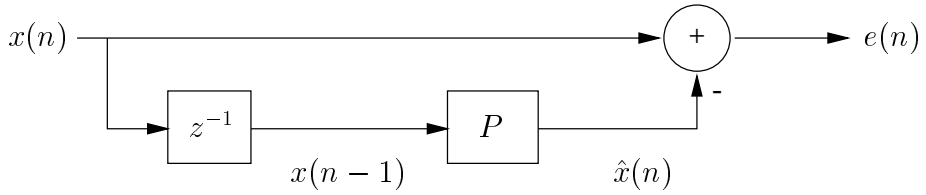
Koodaus voidaan kuvata muodossa

$$e(n) = x(n) - \hat{x}(n), \tag{4.29}$$

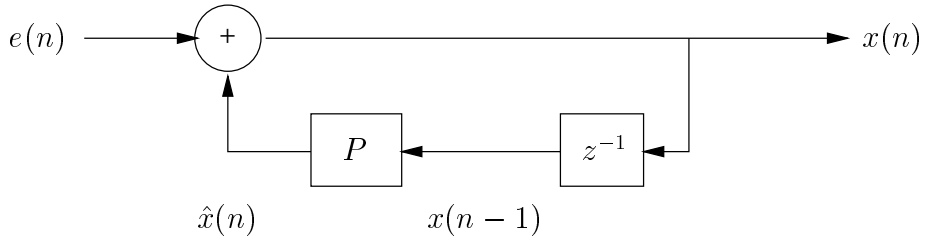
missä $\hat{x}(n)$ on jollain menetelmällä ennustettu $x(n)$ arvo. DPCM-koodaaja on kuvattuna signaalivuokaavion avulla kuvassa 4.1. Alkuperäinen signaali voidaan rekonstruoida täydellisesti virhesignaalista:

$$x(n) = \hat{x}(n) + e(n). \tag{4.30}$$

DPCM-purkaja on kuvattuna kuvassa 4.2. DPCM-järjestelmässä on ennustajan P oltava toiminnaltaan identtinen molemmissä päissä. [11]



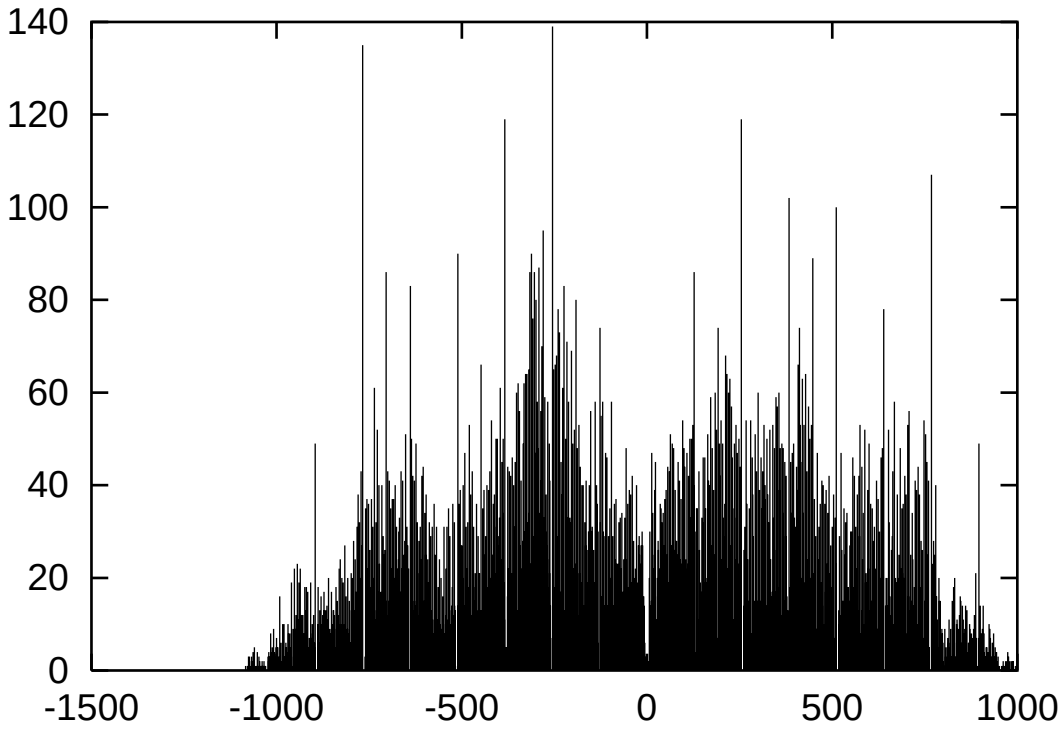
Kuva 4.1: DPCM-koodaaja.



Kuva 4.2: DPCM-purkaja.

Ennustusvirhesignaali voidaan tiivistää tehokkaasti jollain entropian koodauksen menetelmällä. Ennustaja poistaa näytteiden välisen korrelaation ja toimii ti-

lastollisena mallina tiivistysjärjestelmässä. Kuvissa 4.3 ja 4.4 on havainnollistettu tyypillistä eroa alkuperäisen ja ennustusvirhesignaalin jakaumien välillä. Nähdään selvästi, että ennustusvirhe soveltuu paremmin yksinkertaisen entropian koodaajan, kuten Huffman- tai aritmeettinen koodaus, koodattavaksi.



Kuva 4.3: Silmänliikesignaalin arvojen jakauma.

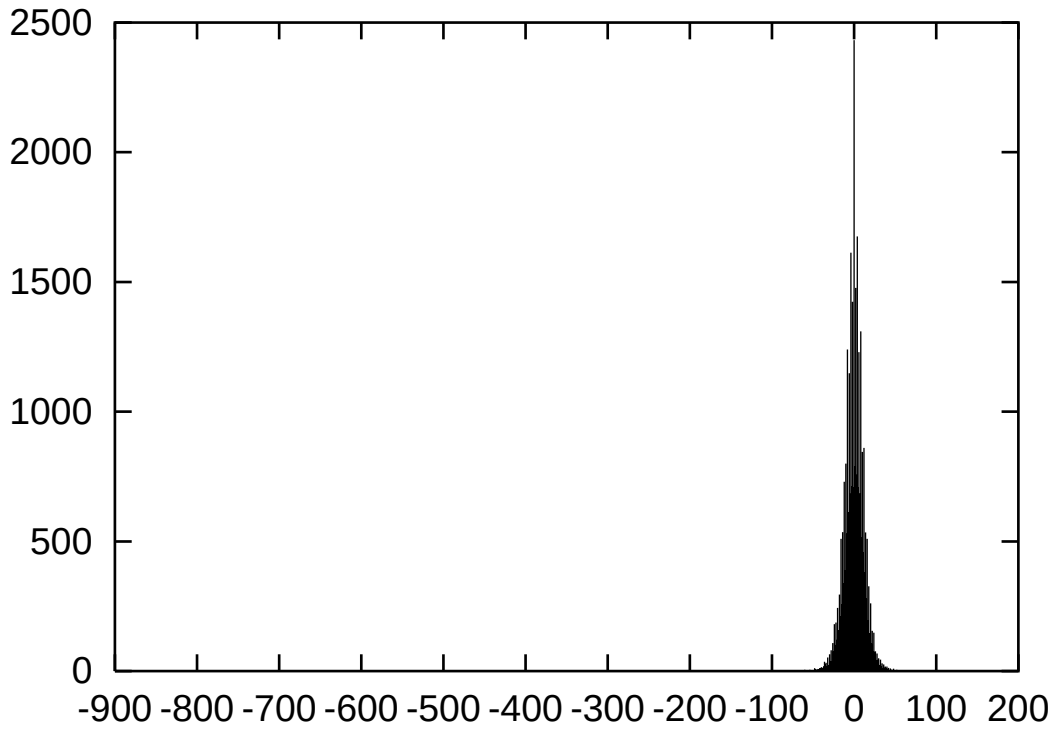
DPCM:n tehokkuus riippuu pääasiassa siitä, miten ennuste $\hat{x}(n)$ muodostetaan. Yksinkertaisin ennustusmenetelmä, nk. *nollannen asteen ennustaja* (zero order predictor, ZOP), ennustaa, että seuraava näyte on sama kuin edellinen.

$$\hat{x}(n) = x(n-1). \tag{4.31}$$

ZOP:ia hieman monimutkaisempi *ensimmäisen asteen ennustaja* (first order predictor, FOP) seuraa suoraa viivaa, eli se olettaa, että muutos pysyy samana näytteiden välillä.

$$\hat{x}(n) = x(n-1) + (x(n-1) - x(n-2)) = 2x(n-1) - x(n-2). \tag{4.32}$$

ZOP ja FOP toimivat hyvin silloin, kun signaali muuttuu hitaasti. On kuitenkin toi-



Kuva 4.4: Ennustusvirhesignaalin arvojen jakauma.

vottavaa löytää menetelmä, jolla voidaan konstruoida sopivia ennustajia tilanteesta riippuen. Voi olla, että tiivistysjärjestelmä joutuu käsittelemään hyvinkin erilaista dataa, jolloin vain yhden kiinteän ennustajan käyttäminen johtaa ongelmiin. Itseasiassa sekä ZOP että FOP voivat huonossa tapauksessa pahentaa tilannetta.

4.3.1 Levinson-Durbin-algoritmi

Levinson ja Durbin kehittivät algoritmin, jolla voidaan muodostaa stationääriselle prosessille MSE-mielessä optimaalinen muotoa

$$\hat{x}(n) = \sum_{m=1}^{M} a_m x(n-m) \tag{4.33}$$

oleva ennustaja sen autokorrelaatiofunktion avulla [28]. M tarkoittaa ennustajan muistin kokoa. Itseasiassa ennustaja saadaan "epäsuorasti" minimoimalla ennustusvirhesuodattimen ulostulon varianssi. MSE-mielessä optimaalista ennustajaa suun-

niteltaessa halutaan minimoida ennustajan tekemä keskineliövirhe:

$$\epsilon = E\left[e^2(n)\right] = E\left[\left(x(n) - \hat{x}(n)\right)^2\right]. \tag{4.34}$$

Lauseke minimoituu, kun sen derivaatat kertoimien suhteen ovat 0, eli

$$\frac{\partial \epsilon}{\partial a_i} = 2E\left[e(n)\frac{\partial e(n)}{\partial a_i}\right] = 0. \tag{4.35}$$

Tästä saadaan yhtälöryhmä, jota kutsutaan *normaaliyhtälöiksi* (normal equations)[3]; sen ratkaisusta saadaan optimaaliset kertoimet ennustajalle

$$E[e(n)x(n-i)] = 0,\ i = 1, 2, \ldots, M. \tag{4.36}$$

Normaaliyhtälöistä on helppo huomata, että havaintojen ja virheen välillä ei saa olla korrelaatiota, kun on kyse optimaalisesta lineaarisesta ennustamisesta. Intuitiivisesti tämä on selvää. Jos niiden välillä olisi korrelaatiota, ennustaja ei käyttäisi kaikkea informaatiota hyväkseen. Tätä kutsutaan *ortogonaalisuusperiaatteeksi*[4].

$$E\left[\left(x(n) - \sum_{j=0}^{M} a_j x(n-j)\right) x(n-i)\right] = 0,\ i = 1, 2, \ldots, M, \tag{4.37}$$

$$r(i) - \sum_{j=1}^{M} a_j r(i-j) = 0,\ i = 1, 2, \ldots, M. \tag{4.38}$$

Normaaliyhtälöt voidaan esittää kompaktisti matriisimuodossa ja ratkaista kääntämällä autokorrelaatiomatriisi $\mathbf{R}_{xx}$:

$$\mathbf{R}_{xx}(M) = \begin{pmatrix} r_{xx}(0) & r_{xx}(1) & r_{xx}(2) & \ldots & r_{xx}(M) \\ r_{xx}(1) & r_{xx}(0) & r_{xx}(1) & & \\ r_{xx}(2) & r_{xx}(1) & r_{xx}(0) & & \\ \vdots & & & \ddots & \\ r_{xx}(M) & & & & r_{xx}(0) \end{pmatrix}, \qquad \mathbf{a} = \begin{pmatrix} 1 \\ -a_1 \\ -a_2 \\ \vdots \\ -a_M \end{pmatrix}. \tag{4.39}$$

---

[3] Yhtälöryhmä tunnetaan myös nimillä *"äärellisen muistin diskreetti Wiener-Hopf-yhtälö"* ja *"Yule-Walker-yhtälöt"*.

[4] Sana "ortogonaalisuus" tulee ennustuksen geometrisesta tulkinnasta, jota ei tässä käsitellä.

Yhtälöt voidaan nyt kirjoittaa muodossa

$$\begin{pmatrix} r_{xx}(0) & r_{xx}(1) & r_{xx}(2) & \dots & r_{xx}(M) \\ r_{xx}(1) & r_{xx}(0) & r_{xx}(1) & & \\ r_{xx}(2) & r_{xx}(1) & r_{xx}(0) & & \\ \vdots & & & \ddots & \\ r_{xx}(M) & & & & r_{xx}(0) \end{pmatrix} \begin{pmatrix} 1 \\ -a_1 \\ -a_2 \\ \vdots \\ -a_M \end{pmatrix} = \begin{pmatrix} \epsilon_M^+ \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \tag{4.40}$$

missä ϵ_M^+ on ennustajan tekemä keskineliövirhe. Yhtälön oikea puoli ylintä alkiota lukuunottamatta on havaintojen ja ennustusvirheen välinen korrelaatio.

Stationäärisen prosessin autokorrelaatiomatriisi $\mathbf{R}_{xx}$ on symmetrinen ja nk. *Toepliz-matriisi* (ts. jokaisella diagonaalilla on sama alkio), jolloin sen rakennetta voidaan käyttää hyväksi yhtälöiden ratkaisussa. Yhtälöiden ratkaisu voidaan rakentaa ratkaisemalla ensin pienemmän muistin ennustaja ja käyttämällä ratkaisua suuremman ennustajan ratkaisun apuna. Ratkaisu voidaan aloittaa seuraavasti:

$$\mathbf{R}_{xx}(2) \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \epsilon_0^+ \\ \epsilon_0^- \end{pmatrix}, \tag{4.41}$$

josta saadaan alkuarvot ϵ_0^+ ja ϵ_0^-:lle. Seuraavaksi ratkaistaan 1 kertoimen ennustaja:

$$\mathbf{R}_{xx}(2) \begin{pmatrix} 1 \\ -a_1(1) \end{pmatrix} = \begin{pmatrix} \epsilon_1^+ \\ 0 \end{pmatrix}. \tag{4.42}$$

Autokorrelaatiomatriisin ominaisuuksien perusteella pitää yleisesti paikkansa, että jos

$$\mathbf{R}_{xx}(M) \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_M \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_M \end{pmatrix}, \tag{4.43}$$

niin

$$\mathbf{R}_{xx}(M)\begin{pmatrix} x_M \\ x_{M-1} \\ \vdots \\ x_0 \end{pmatrix} = \begin{pmatrix} y_M \\ y_{M-1} \\ \vdots \\ y_0 \end{pmatrix}. \tag{4.44}$$

Kokeillaan ratkaisua, joka on muotoa:

$$\begin{pmatrix} 1 \\ -a_1(1) \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} - K_1 \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \tag{4.45}$$

eli

$$\mathbf{R}_{xx}(2)\left[\begin{pmatrix} 1 \\ 0 \end{pmatrix} - K_1 \begin{pmatrix} 0 \\ 1 \end{pmatrix}\right] = \left[\begin{pmatrix} \epsilon_0^+ \\ \epsilon_0^- \end{pmatrix} - K_1 \begin{pmatrix} \epsilon_0^- \\ \epsilon_0^+ \end{pmatrix}\right]. \tag{4.46}$$

K_1 voidaan ratkaista yhtälöstä:

$$\begin{pmatrix} \epsilon_1^+ \\ 0 \end{pmatrix} = \begin{pmatrix} \epsilon_0^+ \\ \epsilon_0^- \end{pmatrix} - K_1 \begin{pmatrix} \epsilon_0^- \\ \epsilon_0^+ \end{pmatrix}. \tag{4.47}$$

Seuraava ratkaisu tarvitsee arvoa ϵ_1^-, joka saadaan yhtälöstä

$$\mathbf{R}_{xx}(3)\begin{pmatrix} 1 \\ -a_1(1) \\ 0 \end{pmatrix} = \begin{pmatrix} \epsilon_1^+ \\ 0 \\ \epsilon_1^- \end{pmatrix}. \tag{4.48}$$

Kahden kertoimen ennustajan ratkaisemiseksi pitää ratkaista yhtälö:

$$\mathbf{R}_{xx}(3)\begin{pmatrix} 1 \\ -a_1(2) \\ -a_2(2) \end{pmatrix} = \begin{pmatrix} \epsilon_2^+ \\ 0 \\ 0 \end{pmatrix}. \tag{4.49}$$

Ratkaisu on muotoa

$$\begin{pmatrix} 1 \\ -a_1(2) \\ -a_2(2) \end{pmatrix} = \begin{pmatrix} 1 \\ -a_1(1) \\ 0 \end{pmatrix} - K_2 \begin{pmatrix} 0 \\ -a_1(1) \\ 1 \end{pmatrix}. \tag{4.50}$$

K_2 voidaan ratkaista yhtälöstä

$$\begin{pmatrix} \epsilon_2^+ \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \epsilon_1^+ \\ 0 \\ \epsilon_1^- \end{pmatrix} - K_2 \begin{pmatrix} \epsilon_1^- \\ 0 \\ \epsilon_1^+ \end{pmatrix}. \tag{4.51}$$

Menettelyä voidaan jatkaa aina haluttuun kertoimien määrään asti. ϵ^+ sisältää aina kunkin kierroksen jälkeen ennustajan tekemän virheen neliön. Sitä voidaan käyttää apuna ennustajaa suunniteltaessa. Kertoimia K_i sanotaan *heijastuskertoimiksi* (reflection coefficient) tai *PARCOR-kertoimiksi*. Ne ilmaisevat näytteiden välisen suoran riippuvuuden. PARCOR-kertoimien avulla voidaan myös toteuttaa ennustusvirhesuodatin nk. *lattice*-muodossa. On huomattava, että matriisiyhtälöissä on esitetty optimaalisen ennustajan kertoimet, mutta algoritmikuvauksessa kertoimet a_i ovat yksinkertaisuuden vuoksi optimaalisen ennustusvirhesuodattimen kertoimet. Ennustajan kertoimet saadaan vaihtamalla kertoimien merkkiä ja jättämällä pois a_0. Ennustusvirhesuodattimen ja ennustajan yhteyttä on havainnollistettu kuvissa 4.5 ja 4.6.

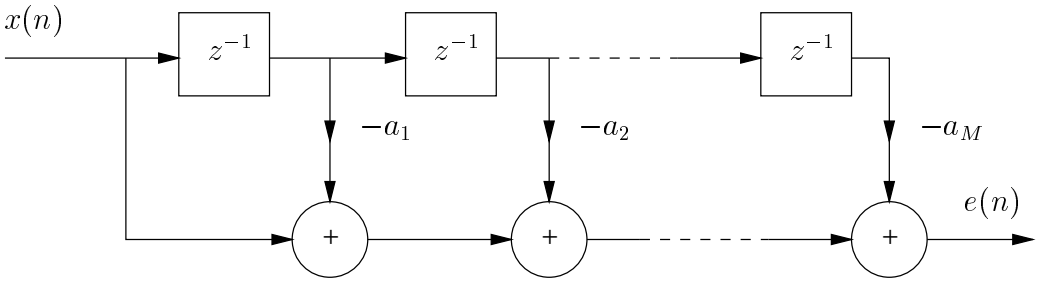
Algoritmi 9 Levinson-Durbin-algoritmi

1: $a_0(0) := 1$ ja $\epsilon_0^+ := r_{xx}(0)$.
2: **for** $m = 0, \ldots, M-1$ **do**
3: $\quad \epsilon_m^- := \sum_{i=0}^{m} a_i(m) r_{xx}(m+1-i)$
4: $\quad K_{m+1} := \epsilon_m^- / \epsilon_m^+$
5: $\quad a_0(m+1) := 1$
6: $\quad$ **for** $i = 1, \ldots, m+1$ **do**
7: $\quad\quad a_i(m+1) := a_i(m) - K_{m+1} a_{m+1-i}(m)$
8: $\quad$ **end for**
9: $\quad \epsilon_{m+1}^+ := (1 - K_{m+1}^2)\epsilon_m^+$
10: **end for**

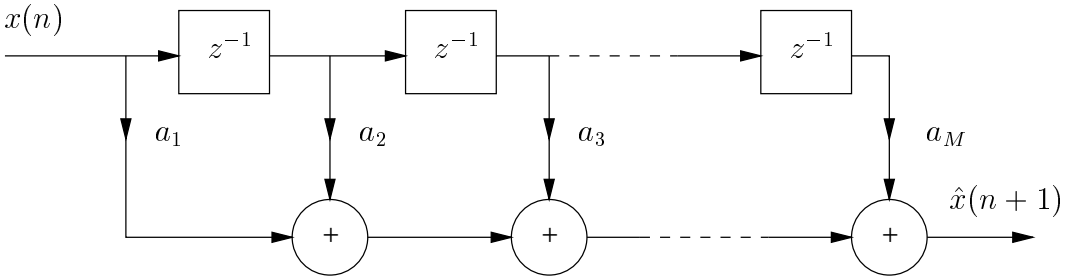
Levinson-Durbin-algoritmilla voidaan siis suunnitella optimaalinen lineaarinen ennustaja stationääriselle prosessille. Käytännön tilanteissa prosessit eivät kuitenkaan usein ole stationäärisiä, jolloin voidaan käyttää mukautuvia ennustajia. Yksinkertaisin ja tunnetuin näistä on LMS-ennustaja.

4.3.2 LMS-algoritmi

Stationääriselle prosessille voidaan suunnitella optimaalinen ennustaja myös opettamalla. Widrowin ja Hopfin vuonna 1960 kehittämä LMS-algoritmi toimii tunnetulla



Kuva 4.5: Ennustusvirhesuodatin.



Kuva 4.6: Ennustaja.

laskeutuvan gradientin menetelmällä. LMS on tunnetuin mukautuvista suodatinalgoritmeista; sitä käytetään mm. neurolaskennassa. [13, 14]

LMS-algoritmin kuvaamiseksi on määriteltävä ϵ:n *gradienttivektori* $\nabla(n)$:

$$\nabla(n) = \begin{pmatrix} \partial e^2(n)/\partial a_1 \\ \partial e^2(n)/\partial a_2 \\ \vdots \\ \partial e^2(n)/\partial a_M \end{pmatrix} = \begin{pmatrix} -2E\left[e(n)x(n-1)\right] \\ -2E\left[e(n)x(n-2)\right] \\ \vdots \\ -2E\left[e(n)x(n-M)\right] \end{pmatrix} = -2E\left[e(n)\mathbf{x}(n-1)\right], \tag{4.52}$$

missä $\mathbf{x}(n-1)$ on ennustajan syötevektori ajan hetkellä n.

$$\mathbf{x}(n-1) = \left(x(n-1), x(n-2), \ldots, x(n-M)\right)^T. \tag{4.53}$$

Merkitään ennustajan kerroinvektoria $\mathbf{a}$:lla, missä

$$\mathbf{a} = (a_1, a_2, \ldots, a_M)^T. \tag{4.54}$$

Ennuste $\hat{x}(n)$ voidaan ilmaista ilmaista tiiviissä muodossa

$$\hat{x}(n) = \mathbf{a}^T \mathbf{x}(n-1). \tag{4.55}$$

Vastaavasti ennustusvirhe $e(n)$ voidaan kirjoittaa seuraavasti:

$$e(n) = x(n) - \mathbf{a}^T(n)\mathbf{x}(n-1). \tag{4.56}$$

Laskeutuvan gradientin menetelmän (gradient descent) ideana on seurata gradientin avulla jotain funktiota minimiinsä. Tässä tapauksessa seurataan ennustajan tekemää virhettä kertoimien funktiona, syntynyttä funktiota kutsutaan *virhepinnaksi*. Virhepinnalla on yksikäsitteinen minimi ja se muistuttaa kulhoa $M+1$-ulotteisessa avaruudessa. Gradienttivektori osoittaa aina funktion suurimpaan kasvuun päin, joten korjataan jokaisella kierroksella paikkaa juuri päinvastaiseen suuntaan; minimi saavutetaan, kun gradienttivektori on 0. Tulkitaan kertoimien arvot pisteeksi M-ulotteisessa avaruudessa. Vektori $\mathbf{a}(n)$ on kertoimien arvo kierroksella n. Kertoimien päivitys voidaan kuvata yhtälöllä

$$\mathbf{a}(n+1) = \mathbf{a}(n) + \frac{1}{2}\mu[-\nabla(n)] = \mathbf{a}(n) + \mu E[e(n)\mathbf{x}(n-1)], \tag{4.57}$$

missä μ on nk. askelkokoparametri tai oppimisnopeusparametri. Parametrin μ arvo vaikuttaa algoritmin konvergoimisnopeuteen. Liian suuri μ:n arvo johtaa ylikorjaukseen (pahimmassa tapauksessa epästabiilisuuteen), liian pieni hitaaseen konvergenssiin. Yhtälöä (4.57) ei voida soveltaa käytännössä sellaisenaan, koska virhepinnan gradienttia ei tunneta. LMS-algoritmissa gradientti ∇(n) korvataan sen hetkellisellä estimaatilla $\hat{\nabla}(n)$, joka määritellään yhtälöllä

$$\hat{\nabla}(n) = -2e(n)\mathbf{x}(n-1). \tag{4.58}$$

Gradientin estimaatin odotusarvo on itse gradientti. LMS-algoritmi voidaan ilmaista muodossa

$$\mathbf{a}(n+1) = \mathbf{a}(n) + \frac{1}{2}\mu[-\hat{\nabla}(n)] = \mathbf{a}(n) + \mu e(n)\mathbf{x}(n-1). \tag{4.59}$$

On osoitettu, että LMS-algoritmi konvergoi (ts. kerroinvektorin ja optimaalisen ker-

roinvektorin välinen etäisyys lähestyy nollaa), jos

$$0 < \mu < \frac{2}{\lambda_{\max}}, \tag{4.60}$$

missä $\lambda_{\max}$ on autokorrelaatiomatriisin $\mathbf{R}_{xx}$ suurin ominaisarvo. [13]

Jos prosessi ei ole stationäärinen, seuraa LMS-algoritmi muuttuvaa virhepintaa yrittäen pysytellä sen minimissä. Optimaalinen kerroinvektori muuttuu ajan mukana ja algoritmin täytyy seurata sitä. Tässä tapauksessa osa algoritmin tekemästä ylimääräisestä virheestä johtuu gradientin kohinasta ja osa siitä, että mukautuva algoritmi on aina hieman optimaalista vektoria jäljessä. [13]

Haykin [13] esittää myös muita versioita LMS-algoritmista. Yksi niistä on Nagumon ja Nodan ja Albertin ja Garderin vuonna 1967 toisistaan riippumatta esittämä normalisoitu LMS, jossa kerroinvektorin päivitys määritellään yhtälöllä

$$\mathbf{a}(n+1) = \mathbf{a}(n) + \frac{\alpha}{\beta + ||\mathbf{x}(n-1)||^2} e(n)\mathbf{x}(n-1), \tag{4.61}$$

missä β on vakio, jolla estetään algoritmin stabiilisuusongelmat, kun vektori $\mathbf{x}(n)$ on pieni. Haykinin[13] mukaan vakion lisäämistä alkuperäiseen NLMS-algoritmiin ehdottivat Bitmead ja Anderson vuonna 1980. Normalisoidulle LMS:lle konvergenssiehto (ilman vakiota β) on yksinkertaisempi:

$$0 < \alpha < 2. \tag{4.62}$$

Tutkimuksessa käytettiin mukautuvana ennustajana myös Least Squares Lattice-algoritmia (LSL), jonka johtamisessa ei ole tehty approksimaatioita. LSL konvergoi erittäin nopeasti verrattuna LMS-algoritmiin ja sen konvergointinopeus ei juurikaan riipu ennustettavan prosessin tilastollisista ominaisuuksista. Algoritmin johto on liian pitkä tässä esitettäväksi; sekä LSL-algoritmin johto että kuvaus löytyvät kirjallisuudesta [1].

4.4 Kvantisointi

Yksinkertaisin häviöllisistä signaalin tiivistyksen menetelmistä on kvantisointi, sitä käytetään mm. mittauksia tehdessä A/D-muunnoksessa. Sitä käytetään myös monimutkaisempien tiivistysjärjestelmien osana. Kvantisoinnin ideana on harventaa signaalin saamien arvojen joukkoa, jolloin se voidaan esittää pienemmällä määrällä

bittejä. Suurin ongelma kvantisoinnissa on valita arvot, jotka poistetaan, ja uudet arvot, joille poistetut arvot kuvautuvat. Tässä käsitellään *skalaarikvantisointia*, jossa syöte ja tulos ovat skalaareita. Skalaarikvantisoinnin voidaan ajatella approksimoivan jotain lukujoukkoa sitä harvemmalla lukujoukolla. Toinen kvantisoinnin muoto, *vektorikvantisointi*, approksimoi vektorijoukkoa harvemmalla vektorijoukolla. Kvantisoijien suunnitteluun käytetty *Lloydin algoritmi* yleistyy varsin siististi myös vektorikvantisointiin. [11]

Määritelmä 4.4.1 N-pisteen skaalarikvantisoija Q on kuvaus $Q : \mathbb{R} \to \mathcal{C}$, missä

$$\mathcal{C} = \{y_1, y_2, y_3, y_4, \ldots, y_N\} \subset \mathbb{R} \tag{4.63}$$

on tulosjoukko tai *koodikirja*, jonka koko $|\mathcal{C}| = N$. Tulosjoukon arvoja y_i kutsutaan myös ulostulotasoiksi, ulostulopisteiksi tai rekonstruktioarvoiksi. Arvot y_i ovat yleensä järjestettyjä siten, että ne ovat indeksin mukaan nousevassa järjestyksessä.[11]

Kvantisoija kuvaa jokaisen syötearvon jollekin koodikirjan arvolle. Skalaarikvantisoijaan kuuluu myös $\mathbb{R}$:n jako N:n soluun tai atomiin R_i, $i = 1, 2, \ldots, N$. Solu määritellään

$$R_i = \{x \in \mathbb{R} : Q(x) = y_i\} = Q^{-1}(y_i). \tag{4.64}$$

Solut jakavat reaalilukujen joukon erillisiin osiin. Kvantisoija on *säännöllinen* (regular), jos

1. jokainen solu R_i on muotoa (x_{i-1}, x_i) mukaanlukien toinen tai molemmat välin päätepisteistä ja

2. $y_i \in (x_{i-i}, x_i)$.

Arvoja x_i kutsutaan *päätöspisteiksi* (decision points). Säännöllisellä kvantisoijalla päätöspisteet ja rekonstruktioarvot ovat järjestettyjä seuraavasti:

$$x_0 < y_1 < x_1 < y_2 < x_2 < \cdots < y_N < x_N. \tag{4.65}$$

Nyt säännöllinen kvantisointi voidaan esittää muodossa

$$Q(x) = y_i, \text{ jos } x_{i-1} < x < x_i. \tag{4.66}$$

Usein ylin ja alin solu ovat nk. *ylikuormitussoluja* (overload cell), jolloin niihin kuuluvat kaikki luvut, jotka ovat solun ylärajaa pienempiä alimman solun tapauksessa ja alarajaa suurempia ylimmän solun tapauksessa. Kvantisoija voidaan myös kuvata kahtena peräkkäisenä kuvauksena: koodaajana $\mathcal{E} : \mathbb{R} \to \mathcal{I}$, missä $\mathcal{I} = \{1, 2, 3, \ldots, N\}$, ja purkajana $\mathcal{D}$, missä $\mathcal{D} : \mathcal{I} \to \mathcal{C}$. Tästä saadaan kvantisointioperaatioksi $Q(x) = \mathcal{D}(\mathcal{E}(x))$. Kvantisoijan suorituskykyä arvioidaan yleensä neliövirheen keskiarvolla, MSE, tai signaali-kvantisointikohina-suhteella, SQNR.

Kvantisoijan suunnitteleminen jollekin datalle, jonka jakauma tunnetaan, vaatii sekä optimaalisen reaalilukujen jaon että optimaalisen rekonstruktiopisteiden suunnittelemisen jonkin metriikan d suhteen. Tämä voidaan tehdä Lloydin algoritmilla (Algoritmi 10), joka sisältää nk. Lloydin iteraation (kohdat 2. ja 3.). Algoritmin kohdassa 2. valitaan soluun ne luvut, joiden etäisyys metriikalla d mitattuna on lyhin solun rekonstruktioarvosta; kohdassa 3. korjataan rekonstruktioarvoja optimaalisiksi soluun nähden. Kohtia 2. ja 3. toistetaan kunnes muutos tulee riittävän pieneksi. [11]

Algoritmi 10 Lloydin algoritmi

1: Aloita jostakin koodikirjasta $\mathcal{C}_1$, $m := 1$.
2: Etsi optimaalinen $\mathbb{R}$:n jako soluihin lähimmän naapurin menetelmällä, ts.

$$R_i = \{x : d(x, y_i) \leq d(x, y_j), \text{ kaikilla } i \neq j\},$$

missä $d(x, y)$ on jokin metriikka.
3: Etsi sellainen $\mathcal{C}_{m+1}$, että jokaista solua vastaava rekonstruktioarvo minimoi keskimääräisen vääristymän.
4: Laske kvantisoijan suorituskyky (esim. MSE). Jos se on muuttunut riittävän vähän edellisestä kierroksesta, lopeta. Jos ei, $m := m + 1$ ja mene kohtaan 2.

Kvantisoija approksimoi reaalilukuja (tai muuta koodikirjaa tiheämpää lukujoukkoa) äärellisellä määrällä lukuja ja on peruuttamaton epälineaarinen operaatio. Tiedon tiivistykseen kvantisointi sopii hyvin., N-pisteen kvantisoijan koodikirjan indeksin esittäminen tarvitsee $\log_2 N$ bittiä. Toisaalta usein kvantisoija joudutaan suunnittelemaan kullekin datalle erikseen, jolloin joudutaan lähettämään vastaanottajalle myös kvantisoijan koodikirja $\mathcal{C}$. Koodikirjan aiheuttama ylimääräinen kustannus on yleensä pieni muuhun dataan verrattuna.

4.5 Häviöllinen DPCM

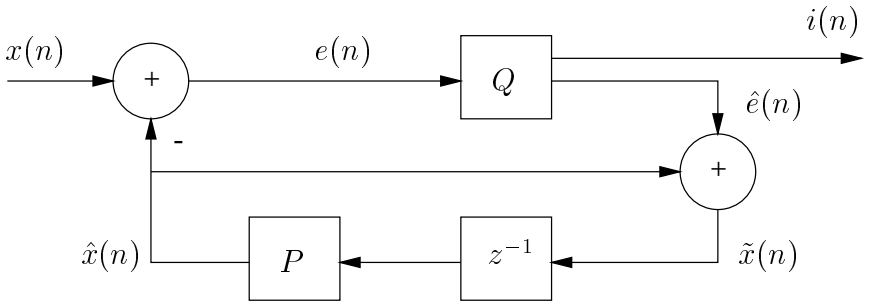
Häviöllisessä DPCM:ssä ennustajan tekemä ennustusvirhe kvantisoidaan ja ennustaja päivitetään aina kvantisoidulla arvolla, jolloin rekonstruoiminen tulee mahdolliseksi. Menetelmällä päästään yleensä laadullisesti parempiin tuloksiin kuin pelkällä kvantisoinnilla. Häviöllinen DPCM-koodaus voidaan kuvata yhtälöillä (4.67)-(4.69):

$$i(n) = \mathcal{E}(x(n) - \hat{x}(n|\tilde{x}(n-1), \tilde{x}(n-2), \ldots, \tilde{x}(n-M))), \tag{4.67}$$

$$\hat{e}(n) = \mathcal{D}(i(n)) \text{ ja} \tag{4.68}$$

$$\tilde{x}(n) = \hat{x}(n) + \hat{e}(n) \tag{4.69}$$

Itse koodaaja on kuvattuna kuvassa 4.7. [11]



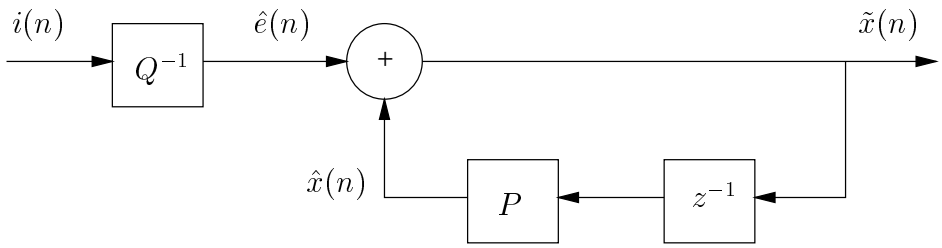
Kuva 4.7: Häviöllinen DPCM-koodaaja.

P on kvantisoinnissa käytetty ennustaja ja vastaanottajalle lähetetään kvantisoijan indeksien muodostama signaali $i(n)$. Vastaavasti rekonstruktio muodostetaan seuraavasti:

$$\tilde{x}(n) = \mathcal{D}(i(n)) + \hat{x}(n|\tilde{x}(n-1), \tilde{x}(n-2), \ldots, \tilde{x}(n-M))). \tag{4.70}$$

DPCM-purkaja on kuvattu kuvassa 4.8.

Vaikka ennustaja käyttääkin ennustuksessa kvantisoituja arvoja $\tilde{x}(n)$, voidaan ennustajana käyttää alkuperäiselle datalle $x(n)$ suunniteltua ennustajaa, koska signaalin $\tilde{x}(n)$ arvojen väliset riippuvuudet ovat suurinpiirtein samat kuin alkuperäisen signaalin. Samoin kvantisoijana voidaan käyttää alkuperäisen signaalin ennustusvirheelle suunniteltua kvantisoijaa, koska kvantisoidusta signaalista ennustava ennustaja tekee suurinpiirtein samanlaisen ennustusvirheen kuin alkuperäiselle signaalille suunniteltu. [11]



Kuva 4.8: Häviöllinen DPCM-purkaja.

Häviöllisellä DPCM:llä päästään hyviin tuloksiin matalilla tiivistyssuhteilla, mutta sillä on vaikea saavuttaa korkeaa laatua suurilla tiivistysuhteilla. Tämä johtuu siitä, että ennustusvirheen kvantisointitasojen määrä tulee liian pieneksi, eikä DPCM pysty enää seuraamaan signaalia riittävän tarkasti, vaan alkaa "heittelehtimään" sen ympärillä.

4.6 Muunnoskoodaus

Muunnoskoodauksella tarkoitetaan menetelmää, jossa signaali muutetaan toiseen ekvivalenttiin esitykseen lohko kerrallaan. Uuden esityksen toivotaan olevan kompaktimpi kuin alkuperäisen. Diskreetissä tapauksessa muunnos voidaan esittää muodossa

$$\mathbf{y} = \mathbf{T}\mathbf{x}, \tag{4.71}$$

missä vektori $\mathbf{y}$ sisältää nk. muunnoskertoimet, $\mathbf{T}$ muunnosmatriisi ja $\mathbf{x}$ alkuperäinen näytevektori. Alkuperäinen näytevektori voidaan palauttaa käänteismuunnoksella:

$$\mathbf{x} = \mathbf{T}^{-1}\mathbf{y}. \tag{4.72}$$

Muunnosmatriisille on tietenkin oltava määritelty käänteismatriisi, jotta alkuperäinen vektori voidaan rekonstruoida.

Muunnos on *ortogonaalinen*, jos

$$\mathbf{T}^{-1} = \mathbf{T}^{*}, \tag{4.73}$$

missä $\mathbf{T}^{*}$ tarkoittaa konjugaattitranspoosia. Jos muunnosmatriisi $\mathbf{T}$ on reaalinen, niin konjugaattitranspoosi vastaa pelkkää transpoosia (ts. $\mathbf{T}^{*} = \mathbf{T}^{T}$). Ortogonaali-

sella muunnoksella on voimassa:

$$||\mathbf{x}||^2 = ||\mathbf{T}\mathbf{x}||^2, \tag{4.74}$$

missä merkintä $||\mathbf{x}||$ tarkoittaa vektorin euklidista pituutta. Ortogonaalisen muunnoksen muunnosmatriisin rivejä kutsutaan kantafunktioiksi ja niiden muodostamaa joukkoa kannaksi. Usein muunnoksesta käytetään nimeä *dekompositio*, jolla tarkoitetaan syötteen jakamista komponentteihin, jotka ovat skaalattuja muunnoksen kantafunktioita. Periaatteessa syötteen voidaan ajatella olevan aluksi esitettynä standardikannalla, joka muodostuu siirretyistä impulssifunktioista.

Mitä tiiviimmin alkuperäisen vektorin informaatio on esitettynä muunnetussa muodossa sen paremmin muunnoskoodaus toimii. Tiivis esitys muunnoskoodauksessa tarkoittaa informaation keskittymistä muunnetun vektorin yksittäisiin alkioihin. Muunnoskoodaus on tavallaan *ad hoc*-menetelmä; kiinteillä kantafunktioilla ei ole mitään takeita siitä, että esitys olisi jotenkin tiivimpi kuin alkuperäinen. Oikeastaan voidaan ajatella, että kyse on tuntemattoman vektorin esittämisestä määrätyn kannan avulla. Ei voida olla varmoja, että kanta on sellainen, joka esittäisi vektorin tiiviisti. Muunnoskoodauksella on kuitenkin muita etuja. Informaatio voidaan saada esitetyksi sellaisessa muodossa, ettei sen karsiminen aiheuta ei-toivottuja artefakteja. Hyvä esimerkki tästä on diskreetti Fourier-muunnos, DFT, jonka avulla voidaan esittää signaali taajuusalueella. Jos tiedetään, mille kaistalle tarvittava informaatio on keskittynyt, voidaan muiden taajuuksien informaatiota karsia.

4.6.1 Tiivistäminen

Muunnoskoodauksessa tiivistys tapahtuu määräämällä bittejä eri tavoin kertoimille; merkisevimmille eniten ja vähemmän merkitseville vähemmän [11]. *Kynnystäminen* (tresholding) on menetelmä, jossa karsitaan muunnoksen kertoimia niiden itseisarvojen suuruuksien perusteella. Kertoimet, joiden itseisarvo on jotain rajaa pienempi, poistetaan kokonaan. Kynnystämisen raja voidaan valita monella eri tavalla. Yksinkertaisin menetelmä on valita vain joku luku, jota itseisarvoltaan pienemmät kertoimet karsitaan. *Kvantiilikynnystämisessä* valitaan kynnystämisen raja dynaamisesti; voidaan esimerkiksi poistaa muunnoksesta itseisarvoltaan pienimmät 75% kertoimista. Kvantiilikynnystämisessä on helppo säädellä tiivistyssuhdetta, mutta ei laatua. [12]

Said ja Pearlman [32] esittävät kuvien tiivistämisessä käytettyjä menetelmiä, joi-

den ideana on koodata asteittain kertoimet merkitsevyyden mukaan järjestyksessä ja kertoimien merkitsevimmät bitit ensin. Koodaus lopetetaan, kun on käytetty määrätty määrä bittejä. Menetelmässä on siis koodattava jokaisella kierroksella aiemmin merkityksettömien kertoimen merkitsevyys ja tarkennettava entisiä merkitseviä kertoimia. Merkitsevyyden koodaamiseen myöhemmin käsiteltävän aallokemuunnosten tapauksessa on kehitetty erilaisia menetelmiä, kuten EZW ja SPIHT [32]. Menetelmiä kutsutaan yleisnimellä *peräkkäisten approksimaatioiden kvantisointi* (successive approximation quantization, SAQ). Etu kynnystämiseen nähden on se, että enemmän bittejä käytetään signaalin kannalta merkitseviin osiin muunnosta; kynnystämisessähän koodataan myös jäljelle jääneiden kertoimien vähiten merkitsevät bitit. Ongelmana on se, että kynnystämisessä valitut komponentit tulevat sellaisenaan rekonstruktioon, mutta tässä kaikkien komponenttien alimmat bitit jäävät yleensä kirjoittamatta, joten rekonstruktiossa komponenttien amplitudi saattaa olla pienempi kuin alkuperäisessä signaalissa. Kehittelin Saidin ja Pearlmanin artikkelissa [32] esitettyjen ideoiden pohjalta menetelmän variantin.

Kehittämäni peräkkäisten approksimaatioiden kvantisoinnin variantti on kuvattu Algoritmissa 11. Algoritmin ideana on kirjoittaa ensin merkitsevimmät bitit ja jatkaa sitten asteittain vähemmän merkitseviin. Kerroin on merkitsevä, jos sen itseisarvo ylittää jonkin rajan. Menetelmässä rajaa lasketaan kierroksittain. Aluksi merkitsevyyden rajaksi asetetaan 2^b (eli x on merkitsevä, jos $|x| \geq 2^b$), missä b on vektorin itseisarvoltaan suurimman kertoimen ylimmän bitin positio. Ensin kaikki kertoimet ovat merkityksettömiä. Ensimmäisellä kierroksella koodataan kaikkien niiden kertoimien merkki ja paikka, jotka ovat merkittäviä. Löydetyt kertoimet poistetaan joukosta ja siirretään merkitsevien listaan. Seuraavilla kierroksilla kirjoitetaan ensin merkitsevien kertoimien seuraava bitti (rivit 5-7), lasketaan merkitsevyyden rajaa bitin verran alemmaksi ja etsitään sitten kertoimet, jotka ovat merkitseviä. Näiden paikat ja merkit koodataan. Paikkojen koodaamiseen käytetään RLE-koodausta, jossa koodataan vuorotellen merkittävien ja merkityksettömien kertoimien juoksuja (rivit 18-21); ensimmäisestä juoksusta on merkittävä tiedostoon, sisältääkö se merkittäviä vai merkityksettömiä kertoimia (rivit 10 ja 16). Juoksujen pituudet esitetään γ-koodilla. Koodataan pituus-1; koska pituus on aina vähintään yksi, olisi nollan koodaaminen redundanttia. Kun huomataan, että juoksu jatkuu listan loppuun asti, voidaan säästää muutama bitti koodaamalla pienin sellaisen γ-koodin alkuosa, joka on suurempi kuin listassa olevien käsittelemättömien kertoimien määrä. Merkitsevyyden koodaaminen voidaan nähdä klusteroituneen bittivektorin koodauk-

sena.

Koodaus voidaan lopettaa, kun haluttu määrä bittejä on käytetty; menetelmä tuottaa siis asteittaisen *upotetun koodin* (embedded code). Koodaajan ja purkajan bittilaskurit on pidettävä synkronoituina, jotta voidaan koodata useita vektoreita samaan tiedostoon. Menetelmän kompleksisuus on $O(mn)$, missä n on koodattavan vektorin pituus ja m vektorin itseisarvoltaan suurimman kertoimen ylimmän bitin positio.

Ero peräkkäisten approksimaatioiden kvantisoinnin variantin ja Saidin esittämän algoritmin [32, algoritmi I] välillä on merkitsevien kertoimien siirtäminen toiseen listaan ja merkitsevien kertoimien tarkentaminen ennen uusien merkitsevien kertoimien etsimistä. Kertoimien siirtämisellä toiseen listaan säästetään tilaa uusien merkitsevien kertoimien sijainnin esityksessä. Kertoimien tarkentamisella ennen uusien merkitsevien bittien etsimistä saadaan viimeiseen bittikerrokseen, jossa käytettävät bitit loppuvat, enemmän bittejä. Said ja Pearlman [32] eivät esitä mitään menetelmää sijainnin koodaamiseen algoritmin yhteydessä; tässä siihen on käytetty RLE-koodausta.

Näiden menetelmien lisäksi voidaan tietysti käyttää myös tavallista kvantisointia, jossa kvantisoija on suunniteltu jokaiselle kertoimelle erikseen ja kvantisoijan tasojen määrä määrätty kertoimen merkitsevyyden perusteella tai vektorikvantisointia.

4.6.2 Diskreetti Fourier-muunnos

Diskreetti Fourier-muunnos (Discrete Fourier Transform, DFT) esittää N-jaksollisen signaalin toistensa kanssa taajuuksiltaan harmonisessa suhteessa olevien siniaalto-komponenttien avulla. Fourier-analyysi on merkittävä menetelmä signaalien analysoinnissa ja DFT-menetelmää käytetään suodattamiseen nk. *nopeassa konvoluutiossa* (fast convolution). Brigham [3] esittää yhteenvedon DFT:n sovelluksista. Tiedon tiivistyksen kannalta DFT ei ole niin tärkeä, koska sen on todettu olevan suorituskyvyltään vaihtoehtojaan heikompi monissa tiedontiivistyssovelluksissa.

Diskreetin N-jaksollisen signaalin x_n diskreetti Fourier-muunnos c_k on

$$X(\frac{2\pi k}{N}) = c_k = \frac{1}{N}\sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N},\ k = 0, \ldots, N-1. \tag{4.75}$$

Yhtälöä kutsutaan myös *analyysiyhtälöksi*. Käänteismuunnoksella kertoimista c_k voi-

Algoritmi 11 Peräkkäisten approksimaatioiden kvantisointi

Alkuehto: x_i on syöte. S on merkitsevien kertoimien lista ja I merkityksettömien kertoimien lista. Kerroin x_i on merkitsevä, jos $|x_i| > 2^b$. Koodaus voidaan lopettaa sopivassa kohdassa, kun haluttu bittien määrä on käytetty.

1: Lisää kertoimet x_i listaan I.
2: $b := \log_2 \max\{|x_i|\} - 1$.
3: Tulosta b 1-bittiä ja 0-bitti.
4: **while** $b \geq 0$ **do**
5: **for all** x_i listassa S **do**
6: Tulosta $|x_i|$:n b:s bitti. {Merkitsevien kertoimien tarkennus.}
7: **end for**
8: **if** $b \geq 0$ **and** lista I ei ole tyhjä **then**
9: **if** listan I ensimmäinen alkio on merkitsevä **then**
10: Tulosta 1-bitti. {Lista I alkaa merkitsevien juoksulla.}
11: Etsi ensimmäisen merkitsevien kertoimien juoksun pituus l.
12: Koodaa $l-1$ γ-koodilla tai, jos juoksu jatkuu listan loppuun asti, koodaa k 1-bittiä, missä $2^k > l$.
13: Koodaa juoksun kerrointen merkkibitit.
14: Siirrä juoksun kertoimet listasta I listaan S.
15: **else**
16: Tulosta 0-bitti. {Lista I alkaa merkityksettömien juoksulla.}
17: **end if**
18: **while** listaa I ei ole käyty läpi **do**
19: Koodaa seuraava merkityksettömien kertoimien juoksun pituus l kuten kohdassa 12.
20: **if** listaa I ei ole käyty läpi **then**
21: Käsittele seuraava merkityksellisten kertoimien juoksu kuten kohdissa 12-14.
22: **end if**
23: **end while**
24: **end if**
25: $b := b - 1$.
26: **end while**

daan rekonstruoida alkuperäinen vektori:

$$x_n = \sum_{k=0}^{N-1} c_k e^{i2\pi kn/N},\ n = 0, \ldots N-1. \tag{4.76}$$

Rekonstruktiota kuvaavaa yhtälöä kutsutaan vastaavasti *synteesiyhtälöksi*. [30]

On huomattava, että edellä olevat yhtälöt on tarkoitettu N-jaksollisille signaaleille. Niitä ei voida sellaisenaan soveltaa mielivaltaisen pituisille signaaleille. DFT-menetelmää voidaan käyttää signaalin analysointiin pilkkomalla se N-pituisiin lohkoihin, jolloin DFT olettaa, että lohko on yksi N-jaksollisen signaalin jakso. Tästä seuraa joitakin ongelmia, joihin palataan myöhemmin. Yhtälöt on esitetty muodossa, jossa ne yleensä esitetään signaalinkäsittelyssä; ne eivät kuvaa ortogonaalista muunnosta. Muunnoksesta saadaan tarvittaessa ortogonaalinen muuttamalla sekä analyysi- että synteesiyhtälöiden vakiokertoimeksi $1/\sqrt{N}$.

DFT tuottaa N-pituisista reaaliarvoisista syötevektoreista N-pituisen kompleksiarvoisen vektorin. Tiedon tiivistyksen kannalta kertoimien kahdentuminen ei ole toivottavaa: Onneksi reaalidatan DFT noudattelee symmetriaa, jonka avulla voidaan poistaa puolet kertoimista. Jos syöte on reaalinen, DFT:llä on voimassa

$$X(\omega) = \mathrm{Re}\left[X(-\omega)\right] - \mathrm{Im}\left[X(-\omega)\right], \tag{4.77}$$

eli kertoimilla on voimassa

$$c_k = c^*_{N-k},\ k = 1, \ldots, N. \tag{4.78}$$

Lisäksi kertoimilla c_0 ja $c_{N/2}$ ei ole imaginääriosaa, kun syöte on reaalinen. [30]

Sellaisenaan DFT:n toteuttaminen vaatisi ajan $O(n^2)$ syötteen pituuden suhteen. Tästä seuraa, että DFT ei sellaisenaan sovi ainakaan reaaliaikasovelluksiin. Cooley ja Tukey julkaisivat vuonna 1965 algoritmin, joka tunnetaan *nopeana Fourier-muunnoksena* (Fast Fourier Transform, FFT). FFT hyödyntää muunnosmatriisin ominaisuuksia ja selviytyy hajoita ja hallitse -algoritmilla tehtävästä ajassa $O(n \log n)$. Sovelluksien kannalta nopeuden lisäys on merkittävä. Käytännössä FFT on mahdollistanut DFT:n käytön reaaliaikaisissa sovelluksissa. Myöhemmin on kehitetty erilaisia FFT:n variaatioita[5], jotka toimivat käytännössä (mutta eivät kuitenkaan kompleksisuudeltaan) nopeammin kuin alkuperäinen FFT. Nopeita FFT-algoritmeja

[5] mm. Radix-4, Split-Radix FFT (SRFFT)

on käsitelty kattavasti kirjallisuudessa [5, 11, 26, 30]. Yleensä FFT toimii vain vektoreille, joiden koko on jotenkin rajattu; yleisin rajoitus on se, että vektorin koon on oltava kahden potenssi. [3]

4.6.3 Hartley-muunnos

Hartley-muunnos (Hartley Transform) on muokattu DFT, joka toimii vain reaaliluvuille [40]. Diskreetin signaalin x_n Hartley-muunnos on

$$c_k = \frac{1}{N} \sum_{n=0}^{N-1} (\cos \frac{2\pi nk}{N} - \sin \frac{2\pi nk}{N}) x_n. \tag{4.79}$$

Vastaavasti alkuperäinen signaali voidaan rekonstruoida muunnoksesta

$$x_n = \sum_{n=0}^{N-1} (\cos \frac{2\pi kn}{N} - \sin \frac{2\pi kn}{N}) c_k. \tag{4.80}$$

Hartley-muunnoksen aikavaatimus nopealla hajoita ja hallitse-algoritmilla toteutettuna on $O(n \log n)$.

4.6.4 Diskreetti kosinimuunnos

Gershon ja Grayn [11] mukaan tiedon tiivistyksessä ehkä käytetyin muunnos on Ahmedin, Natarajanin ja Raon vuonna 1974 kehittämä *diskreetti kosinimuunnos* (Discrete Cosine Transform, DCT). Sitä käytetään mm. tunnetuissa kuvan ja videon tiivistysmenetelmissä JPEG ja MPEG. DCT:n muunnettu versio (Modified Discrete Cosine Transform, MDCT) on käytössä tunnetussa MPEG audio layer 3-tekniikassa.

Diskreetin signaalin x_n diskreetti kosinimuunnos c_k on

$$c_k = \begin{cases} \sqrt{\frac{1}{N}} \sum_{n=0}^{N-1} x_n, \; k = 0 \\ \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} x_n \cos \frac{k(2i+1)\pi}{2N}, \; k = 1, \ldots, N-1 \end{cases} . \tag{4.81}$$

Vastaavasti käänteismuunnos on

$$x_k = \sqrt{\frac{1}{N}}\, c_0 + \sqrt{\frac{2}{N}} \sum_{i=1}^{N-1} c_i \cos \frac{i(2k+1)\pi}{2N}. \tag{4.82}$$

Naiivi DCT-toteutus toimii ajassa $O(n^2)$, mutta DCT voidaan laskea FFT:n avulla ja sille voidaan myös johtaa nopea algoritmi, jolloin muunnoksen aikavaatimus

on $O(n \log n)$. [37]

4.6.5 Walsh-Hadamard-muunnos

Walsh-Hadamard-muunnos (Walsh-Hadamard Transform, WHT) on varsin yksinkertainen; sen muunnosmatriisi voidaan määritellä rekursiivisesti:

$$\mathbf{T}_{\mathrm{WHT}}(0) = 1, \tag{4.83}$$

$$\mathbf{T}_{\mathrm{WHT}}(n+1) = \begin{pmatrix} \mathbf{T}_{\mathrm{WHT}}(n) & \mathbf{T}_{\mathrm{WHT}}(n) \\ \mathbf{T}_{\mathrm{WHT}}(n) & -\mathbf{T}_{\mathrm{WHT}}(n) \end{pmatrix}. \tag{4.84}$$

Vektorin $\mathbf{x}$ Walsh-Hadamard-muunnos $\mathbf{y}$ on muotoa

$$\mathbf{y} = \frac{1}{N}\mathbf{T}_{\mathrm{WHT}}(n)\mathbf{x}, \tag{4.85}$$

missä vektorin $\mathbf{x}$ pituus $N = 2^n$. Vastaavasti alkuperäinen vektori voidaan rekonstruoida:

$$\mathbf{x} = \mathbf{T}_{\mathrm{WHT}}(n)\mathbf{y}. \tag{4.86}$$

WHT esittää signaalin neliöaaltokomponenteilla, joilla on vaihteleva määrä *nollanylityksiä* (zero crossing) [37]. Nopealla algoritmilla WHT voidaan laskea ajassa $O(n \log n)$; algoritmi muistuttaa hyvin läheisesti radix-2 FFT-algoritmia. Esitän nopeista algoritmeista vain johtamani nopean Walsh-Hadamard-muunnoksen paikallaan (in-place). Muiden muunnosten nopeat versiot on esitetty kirjallisuudessa [3, 37, 40].

4.6.6 Aalloke- ja aaltopaketti-muunnokset

Aiemmille muunnoksille on yhteistä se, että niissä jokainen kerroin vaikuttaa jokaiseen rekonstruoituun näytteeseen. Tästä seuraa, että kertoimen muutos näkyy jokaisessa rekonstruoidussa näytteessä. Diskreetin Fourier-muunnoksen ongelma on se, että se erottaa huonosti ilmiöitä, joiden kesto on pienempi kuin ikkunan pituus. *Aallokkeet* (wavelets) voivat analysoida signaaleja monella eri skaalalla; aallokeanalyysissä taajuusalueen ja aika-alueen erottelukyky vaihtelee skaalan mukaan. Matalataajuisten tapahtumien taajuus erotetaan tarkasti, mutta tapahtuma-aika epätarkasti. Korkeataajuiset tapahtumat erotetaan ajan suhteen tarkasti, mutta niiden taajuus epätarkasti. Taajuuden ja ajan välinen epätarkkuus vastaa Heisenbergin

Algoritmi 12 Nopea Walsh-Hadamard-muunnos paikallaan

Alkuehto: $\mathbf{x} = (x_0, x_1, \ldots, x_{N-1})$ on muunnettava data.
Jättöehto: $\mathbf{x}$ sisältää datan skaalaamattoman Walsh-Hadamard-muunnoksen.

```
1: koko := log_2(N) − 1.
2: for i := 0 to koko do
3:    for j := 0 to 2^koko step 2^(i+1) do
4:       for k := 0 to 2^i do
5:          t := x_(j+k).
6:          x_(j+k) := t + x_(j+k+2^i).
7:          x_(j+k+2^i) := t − x_(j+k+2^i).
8:       end for
9:    end for
10: end for
```

epätarkkuusperiaatetta. [12]

Aallokemuunnosten teoria on saanut alkunsa 1940-luvulla; Gabor julkaisi vuonna 1945 artikkelin, jossa on aallokkeiden teorian perusta. Käytännön tietoteknisissä sovelluksissa aallokkeita on käytetty vasta 1980-luvulla. Aallokemuunnosten etu perinteisiin muunnoksiin nähden on se, että ne esittävät funktion sekä taajuus- että aika-alueella lokalisoituneiden funktioiden avulla. Tästä seuraa, että aallokemuunnosten kertoimet eivät kaikki vaikuta jokaiseen näytteeseen rekonstruktiossa. Toisaalta yksittäiset lyhytkestoiset tapahtumat voidaan esittää taloudellisesti pienellä määrällä kertoimia. [12]

Aallokemuunnoksessa funktio esitetään yhden funktion Φ, *äitiaallokkeen* (mother wavelet), translaatioiden ja dilataatioiden muodostaman kannan avulla:

$$\Phi_{(s,l)}(x) = 2^{-s/2}\Phi(2^{-s}x - l), \tag{4.87}$$

missä s on aallokkeen leveys ja l sen paikka. Sekä s että l ovat molemmat kokonaislukuja. Äitiaalloketta siirretään kokonaisluvuilla ja skaalataan kahden potensseilla. Aallokkeiden teoriaa ei tässä tarkemmin tarkastella. Aalloke voidaan esittää nk. aallokesuodattimien avulla. Alipäästösuodatin H tuottaa N-pituisesta syötesignaalista $N/2$-pituisen signaalin, joka sisältää syötesignaalin yleispiirteet. Ylipäästösuodatin G tuottaa myös $N/2$-pituisen signaalin, joka sisältää syötesignaalin yksityiskohdat. Kummassakin tapauksessa suodattimeen sisältyy myös näytteenottotaajuuden laskuoperaatio, joka voidaan tehdä menettämättä informaatiota, koska suodattimet puolittavat signaalin kaistanleveyden. Suodattimet voidaan kuvata kertoimilla h_i ja

g_i, jotka liittyvät toisiinsa kaavalla

$$g_i = (-1)^{M-i} h_{M-i}, \tag{4.88}$$

missä M on suodattimen pituus. Jatkossa signaalit esitetään vektoreina, koska niiden avulla on helpompi havainnollistaa aallokedekomposition toimintaa. [12]

Diskreettiä aallokemuunnosta kutsutaan myös *multiresoluutioanalyysiksi*. Menetelmän ja algoritmin kehitti Mallat vuonna 1989 [12]. Algoritmi tunnetaan yleisesti "pyramidialgoritmina". Määritellään aallokemuunnoksen $N \times N$-muunnosmatriisi $\mathbf{T}_{\mathrm{DWT}}(N)$ seuraavasti:

$$\mathbf{T}_{\mathrm{DWT}}(N) = \begin{pmatrix} h_0 & h_1 & h_2 & \dots & h_M & 0 & \dots & & & 0 \\ g_0 & g_1 & g_2 & \dots & g_M & 0 & \dots & & & 0 \\ 0 & 0 & h_0 & h_1 & h_2 & \dots & h_M & 0 & \dots & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & \dots & g_M & 0 & \dots & 0 \\ \vdots & & & & & & & \ddots & & \\ h_2 & \dots & h_M & 0 & & \dots & & 0 & h_0 & h_1 \\ g_2 & \dots & g_M & 0 & & \dots & & 0 & g_0 & g_1 \end{pmatrix}, \tag{4.89}$$

missä N on kahden potenssi. Määritelmässä on käytetty jaksollista signaalin mallia. Suodattimien kertoimet kiertävät signaalin lopun ympäri takaisin signaalin alkuun siinä tapauksessa, että ne menevät lopun yli. Jaksollisen signaalin mallin käyttö aiheuttaa omat ongelmansa, joihin palataan myöhemmin. Aallokemuunnoksen yksi kierros voidaan nyt esittää muodossa:

$$\mathbf{T}_{\mathrm{DWT}}(N)\,\mathbf{x} = \begin{pmatrix} s_0 \\ d_0 \\ s_1 \\ d_1 \\ \vdots \\ s_{N/2} \\ d_{N/2} \end{pmatrix}. \tag{4.90}$$

Yhtälön oikealta puolelta saadaan kaksi signaalia:

$$\mathbf{s}_{N/2} = (s_0, s_1, \ldots, s_{N/2})^T, \text{ ja} \tag{4.91}$$

$$\mathbf{d}_{N/2} = (d_0, d_1, \ldots, d_{N/2})^T. \tag{4.92}$$

Muunnoksen seuraava kierros suoritetaan vektorille $\mathbf{s}_{N/2}$:

$$\mathbf{T}_{\mathrm{DWT}}(N/2)\mathbf{s}_{N/2} = \begin{pmatrix} ss_0 \\ ds_0 \\ ss_1 \\ ds_1 \\ \vdots \\ ss_{N/4} \\ ds_{N/4} \end{pmatrix}, \tag{4.93}$$

josta saadaan taas kaksi uutta signaalia. Menettelyä jatketaan aina alipäästösignaalille $\mathbf{s}$ haluttuun resoluutioon asti. Yleensä lopetetaan, kun tulossignaalien pituus on yksi. [29]

Käänteismuunnos tehdään vastaavasti; muunnosmatriisin transpoosi on sen käänteismatriisi. Alipäästösuodatettuun signaaliin viitataan nimellä s, kahdesti alipäästösuodatettuun nimellä ss, ylipäästösuodatettuun nimellä d, jne. Kahdeksan luvun mittaisen vektorin aallokemuunnos voidaan kuvata seuraavasti:

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \overset{1}{\rightarrow} \begin{pmatrix} s_0 \\ d_0 \\ s_1 \\ d_1 \\ s_2 \\ d_2 \\ s_3 \\ d_3 \end{pmatrix} \overset{2}{\rightarrow} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ \underline{s_3} \\ d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} \overset{1}{\rightarrow} \begin{pmatrix} ss_0 \\ ds_0 \\ ss_1 \\ \underline{ds_1} \\ d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} \overset{2}{\rightarrow} \begin{pmatrix} ss_0 \\ \underline{ss_1} \\ ds_0 \\ \underline{ds_1} \\ d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} \overset{1}{\rightarrow} \begin{pmatrix} \underline{sss_0} \\ \underline{dss_0} \\ ds_0 \\ \underline{ds_1} \\ d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} \tag{4.94}$$

1 Alipäästösignaalin kertominen matriisilla $\mathbf{T}_{\mathrm{DWT}}$.

2 Permutointi, lajitellaan ali- ja yläpäästösignaalit erilleen.

Aallokemuunnos on erittäin nopea; sen kompleksisuus on lineaarinen syötteen koon suhteen. Nopein kaikista aallokemuunnoksista on nk. Haar-muunnos, jossa käyte-

tään kahden näytteen mittaista aallokesuodatinta. [29]

Aaltopakettimenetelmissä (wavelet packet) muunnosta jatketaan haluttuun resoluutioon asti. Erotuksena aallokemuunnokseen on se, että molempien sekä ylieettä alipäästösignaalien muunnosta jatketaan rekursiivisesti ja jokainen välitulos säilytetään. Välitulosten joukosta yritetään etsiä additiivisten informaatiokustannusfunktioiden avulla ne, joissa informaatio on tiiviimmillään. Etsitään siis parasta muunnosta useiden mahdollisuuksien joukosta. Muunnosten kannat koostuvat aallokkeiden lineaarikombinaatioista. [42]

Menetelmän toiminta perustuu siihen, että signaali s hajotetaan suodattamalla yhdellä kierroksella signaaleiksi ss ja ds ja se voidaan rekonstruoida signaaleista ss ja ds. Signaalit ss ja ds voidaan edelleen rekursiivisesti rekonstruoida signaaleista, jotka on saatu niitä hajottamalla. Menetelmässä yritetään löytää informaation taloudellisen esittämisen kannalta paras kombinaatio suodatuksia. Aallokesuodattimista voidaan siis johtaa useita erilaisia muunnoksia ja varsinainen aallokemuunnos onkin vain yksi erikoistapaus. Joitakin vaihtoehtoisia aallokesuodattimesta johdettuja kantoja $\mathbb{R}^8$:lle on esitettynä kuvissa 4.10-4.12. [42]

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
s_0	s_1	s_2	s_3	d_0	d_1	d_2	d_3
ss_0	ss_1	ds_0	ds_1	sd_0	sd_1	dd_0	dd_1
sss_0	dss_0	sds_0	dds_0	ssd_0	dsd_0	sdd_0	ddd_0

Kuva 4.9: Muunnosten hierarkia $\mathbb{R}^8$:lle.

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
s_0	s_1	s_2	s_3	d_0	d_1	d_2	d_3
ss_0	ss_1	ds_0	ds_1	sd_0	sd_1	dd_0	dd_1
sss_0	dss_0	sds_0	dds_0	ssd_0	dsd_0	sdd_0	ddd_0

Kuva 4.10: Aallokemuunnoksen kanta.

Tehtävänä on nyt löytää jokin menetelmä, jolla voidaan mitata mahdollisten kantojen paremmuutta ja algoritmi, jolla voidaan valita paras kanta menetelmän perusteella. Vertailuun käytetään nk. *additiivista informaatiokustannusfunktiota* (additive information cost function).

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
s_0	s_1	s_2	s_3	d_0	d_1	d_2	d_3
ss_0	ss_1	ds_0	ds_1	sd_0	sd_1	dd_0	dd_1
sss_0	dss_0	sds_0	dds_0	ssd_0	dsd_0	sdd_0	ddd_0

Kuva 4.11: Alikaistakanta (subband basis).

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
s_0	s_1	s_2	s_3	d_0	d_1	d_2	d_3
ss_0	ss_1	ds_0	ds_1	sd_0	sd_1	dd_0	dd_1
sss_0	dss_0	sds_0	dds_0	ssd_0	dsd_0	sdd_0	ddd_0

Kuva 4.12: Jokin nimetön esimerkkikanta.

Määritelmä 4.6.1 Olkoon $\{x_i\}$ reaalilukujono. $\mathcal{M} : \{x_i\} \to \mathbb{R}$ on additiivinen informaatiokustannusfunktio, jos $\mathcal{M}(0) = 0$ ja $\mathcal{M}(\{x_i\}) = \sum_i \mathcal{M}(x_i)$.

Wickerhauser[42] esittää vaihtoehdoiksi mm.

1. Tietyn rajan ϵ ylittävien arvojen lukumäärä, jolloin saadaan kertoimien määrä, jolla voidaan esittää signaali tarkkuudella ϵ.

2. Keskittyminen ℓ^p normissa, $p < 2$. $\mathcal{M}(\{x_i\}) = ||\{x_i\}||_p$.

3. Entropia. $\mathcal{M}(\{x_i\}) = -\sum_i |x_i|^2 \log |x_i|^2$, missä tarvittaessa $0 \log 0 = 0$.

4. Energian logaritmi. $\mathcal{M}(\{x_i\}) = \sum_i \log |x_i|^2$, missä tarvittaessa $\log 0 = 0$.

Informaatiokustannuksen $\mathcal{M}$ suhteen optimaalinen kanta (ts. kanta, joka minimoi kustannuksen) saadaan yksinkertaisella algoritmilla. Muunnosten hierarkia muodostaa täydellisen binääripuun, jossa jokaiseen solmuun liittyy vaihtelevan mittainen signaali. Siirtyminen vanhemmasta lapsisolmuun vastaa yhtä aallokemuunnoksen kierrosta; toinen lapsisolmuista sisältää alipäästösignaalin ja toinen ylipäästösignaalin. Puun juurisolmussa on alkuperäinen signaali. Aloitetaan alimmista solmuista ja lasketaan niiden informaatiokustannus ja merkitään alimmat solmut. Mennään ylemmälle tasolle ja lasketaan sen solmujen informaatiokustannus. Jos solmun

informaatiokustannus on pienempi kuin sen lapsisolmujen yhteenlaskettu informaatiokustannus, merkitään solmu. Asetetaan solmun kustannukseksi minimi sen lasten yhteenlasketusta kustannuksesta ja sen omasta kustannuksesta. Menettelyä jatketaan, kunnes tullaan puun juureen. Optimaaliseen kantaan kuuluvat puussa ylimmät merkityt solmut. Aikavaatimukseksi aaltopakettimuunnokselle saadaan $O(n \log n)$. [42]

4.6.7 Muunnoskoodauksen ongelmia

Tyypillinen ongelma muunnoskoodauksessa on jaksollisen signaalin mallin käytön aiheuttamat reunaepäjatkuvuudet. Selkeimmin tämä näkyy muunnoksilla, joiden kannat ovat tasaisia, kuten Fourier-muunnos. Jos muunnoslohkon päät ovat selvästi eri tasoilla, luulee muunnos, että lohkon päädyn kohdalla on signaalissa epäjatkuvuus. Tällaisen epäjatkuvuuden esittäminen jatkuvilla funktioilla ei ole taloudellista, vaan epäjatkuvuus leviää useisiin muunnoskertoimiin. Kun kertoimia karsitaan, niin epäjatkuvuuden esitys huononee ja lopulta rekonstruoidun lohkon päädyt lähestyvät toisiaan. Ongelman ratkaisemiseksi voidaan esimerkiksi ottaa muunnos suuremmasta lohkosta ja koodata vain osa lohkosta, jolloin epäjatkuvuudet jäävät lohkon ulkopuolelle.

5 Tuloksia

5.1 Testien toteutuksesta

Toteutin tiivistysalgoritmit C++-ohjelmointikielellä [36] käyttäen apuna standardia kaavainkirjastoa, joka osoittautui erittäin hyödylliseksi. En käyttänyt verkosta saatavilla olevaa valmiita toteutuksia, koska se olisi vaikeuttanut menetelmien muuttamista ja hybridien rakentamista. Osa valmiista toteutuksista oli suunniteltu käyttämään alkion kokona tavua (8 bittiä), kun tässä tutkimuksessa tarvittiin sanaa (16 bittiä) käyttäviä tiivistysmenetelmiä.

Yritin sijoittaa kaikki tiivistysmenetelmät saman sovelluskehyksen sisälle, jolloin niistä saadut tulokset ovat mahdollisimman vertailukelpoisia. Jälkikäteen arvioituna olisi ehkä ollut parempi käyttää jotain interaktiivisempaa ja dynaamista kieltä kehitykseen, koska tavoitteena ei ollut laskennallinen nopeus, vaan useiden eri menetelmien vertailu ja uusien ideoiden kokeilu. Tässä tapauksessa pääpaino olisi ollut algoritmien toteutuksen vaatimassa ajassa ja toteutuksen virheettömyydessä. Tulosten mittaamisen automatisoin täysin inhimillisen virheen mahdollisuuden minimoimiseksi.

5.2 Häviöttömät menetelmät

Häviöttömistä menetelmistä on ilmoitettu niiden saavuttaman tiivistyssuhteen keskiarvo μ_{CR} ja tiivistyssuhteen keskihajonta σ_{CR}. Osa häviöttömien menetelmien tuloksista on esitetty aiemmin kirjallisuudessa [39]; pienet erot tulosten välillä johtuvat siitä, että joitakin menetelmiä on myöhemmin hieman parannettu.

Ennustusmenetelmiä vertailin ennustusvirheen MAE:n ja MSE:n avulla. Tuloksissa niistä on esitetty keskiarvot μ_{MAE}, μ_{MSE} ja keskihajonnat σ_{MSE}, σ_{MAE}. Taulukossa 5.1 esitetyt tulokset on mitattu signaaleilla, jotka olivat suodattamattomia. Suodatetuilla signaaleilla ennustajien tulosten välillä oli myös vain pieni ero. Ennustajien tekemän virheen itseisarvon keskiarvo vaihteli välillä 9-11. Suodatetuilla signaaleilla ZOP:n tulos oli 9.6, jolloin voidaan sanoa, ettei mukautuvilla ennustajilla tai Levinson-Durbin-algoritmilla saavuteta merkittävää etua suodatettujen signaalien tapauksessa ZOP:iin nähden. Tuloksista nähdään, että yksinkertainen ZOP soveltuu näille signaaleille hyvin ja että monimutkaisempien ennustajien käytöstä ei saada kovin suurta etua, mutta ne lisäävät laskennan kompleksisuutta merkittävästi. Esimerkiksi LSL-ennustajan käyttäminen aritmeettisen koodaajan kanssa paransi keskimääräistä tiivistyssuhdetta noin prosentilla ZOP:n käyttöön verrattuna. Kes-

Ennustaja	μ_{MAE}	σ_{MAE}	μ_{MSE}	σ_{MSE}
ZOP	18.9	18.9	1416	2730
FOP	27.7	29.7	3148	6365
LD(1)	18.9	18.8	1412	2719
LD(3)	18.3	17.7	1279	2413
LD(5)	17.3	16.1	1146	2227
NLMS(1)	20.5	19.4	1788	2890
NLMS(3)	20.4	18.6	1616	2716
NLMS(5)	19.9	17.6	1523	2620
NLMS(7)	19.7	17.4	1491	2593
NLMS(15)	18.8	16.5	1381	2490
LSL(1)	19.3	18.9	1496	2735
LSL(3)	18.3	17.7	1354	2435
LSL(5)	17.1	16.1	1208	2265
LSL(7)	16.7	15.8	1168	2226
LSL(15)	15.9	15.0	1089	2139

Taulukko 5.1: Ennustajien tulokset.

kimääräinen ennustusvirhe putoaa hitaasti, kun lisätään uusia näytteitä ennustukseen ja mukautuvasta ennustuksesta ei juuri ole hyötyä verrattuna staattiseen ennustukseen. Jatkossa DCPM:n kohdalla on käytetty vain ZOP-ennustajaa.

Käytetyt universaalikoodit sopivat hyvin jakaumille, joissa todennäköisyys laskee luvun itseisarvon kasvaessa. Lähteen jakauma DPCM:n jälkeen on juuri tällainen. Ennen ennustusta jakauma on lähes tasainen, jolloin nämä koodit eivät toimi. Lukuja ei järjestetty todennäköisyyden mukaan ennen koodausta, koska ne olivat ennustuksen jälkeen lähes järjestyksessä (ks. kuvat 4.3, s. 44, ja 4.4, s. 45). Tulokset universaalikoodeilla on esitetty taulukossa 5.2.

Menetelmä	Suodatettu	Ennustaja	μ_{CR}	σ_{CR}
γ-koodi		ZOP	1.79	0.50
exp-Golomb-koodi (k=2)		ZOP	2.08	0.54
γ-koodi	x	ZOP	2.31	0.82
exp-Golomb-koodi (k=2)	x	ZOP	2.45	0.54

Taulukko 5.2: Universaalikoodien tuloksia.

Sanakirjamenetelmillä oli suuren aakkoston takia vaikeuksia löytää sovituksia. Toisaalta LZW toimi hyvin ilman DPCM:a; ilmeisesti syynä oli se, että LZW rakentaa koodauksen aikana eräänlaisen yksinkertaisen tilastollisen mallin. LZSS:ssä

käytettiin sovituksen pituuden ylärajana 64 merkkiä ja taaksepäin-puskurin pituus oli 1024 merkkiä. Ilman DPCM:a olivat tulokset melko huonoja; tästä voidaan todeta, etteivät yleiskäyttöiset sanakirjapohjaiset tiivistysmenetelmät toimi signaaleilla kovin hyvin. Sanakirjamenetelmien tulokset on esitetty taulukossa 5.3.

Menetelmä	Suodatettu	Ennustaja	μ_{CR}	σ_{CR}
LZSS		-	0.98	0.06
LZSS		ZOP	1.02	0.13
LZSS	x	-	1.03	0.12
LZSS	x	ZOP	1.14	0.14
LZW		-	1.22	0.21
LZW		ZOP	1.87	0.57
LZW	x	-	1.27	0.25
LZW	x	ZOP	2.26	0.7

Taulukko 5.3: Sanakirjamenetelmien tulokset.

Sekä Huffman- että aritmeettinen koodaus toimivat hyvin, kun DPCM oli käytössä. Menetelmät mallinsivat näytteen 0.:n asteen lähteenä, jolloin ne tarvitsivat hyvän tiivistystuloksen saavuttamiseksi erillisen tilastollisen mallin, joka oli tässä tapauksessa DPCM:n ennustaja. Tulokset luultavasti kärsivät osittain siitä, että pitää tallettaa Huffman-koodin tapauksessa koodipuu ja aritmeettisen koodin tapauksessa välien jako tiedostoon. Signaalit olivat lyhyitä, joten tämä ylimääräinen kustannus saattaa olla merkittävä. Pidemmillä signaaleilla menetelmät olisivat todennäköisesti saavuttaneet hieman parempia tuloksia. Menetelmien tulokset on esitetty taulukossa 5.4.

Menetelmä	Suodatettu	Ennustaja	μ_{CR}	σ_{CR}
Huffman		-	1.14	0.18
Huffman		ZOP	2.27	0.57
Huffman	x	-	1.10	0.17
Huffman	x	ZOP	2.65	0.71
Aritmeettinen koodaus		-	1.22	0.13
Aritmeettinen koodaus		ZOP	2.29	0.53
Aritmeettinen koodaus	x	-	1.19	0.12
Aritmeettinen koodaus	x	ZOP	2.68	0.71

Taulukko 5.4: Huffman- ja aritmeettisen koodauksen tulokset.

Mukautuvat universaalikoodit pystyvät mukautumaan ennustajien tekemien vir-

heiden suuruuden paikallisiin muutoksiin, jolloin saavutetaan varsin hyviä tuloksia. Lisäksi menetelmät ovat toteutukseltaan yksinkertaisia ja laskennallisesti tehokkaita. Testeissä puskurin kokona käytettiin 50 näytettä. Mukautuvien universaalikoodien tulokset on esitetty taulukossa 5.5.

Menetelmä	Suodatettu	Ennustaja	μ_{CR}	σ_{CR}
Mukautuva exp-Golomb		-	1.13	0.09
Mukautuva exp-Golomb		ZOP	2.23	0.53
Mukautuva exp-Golomb	x	-	1.14	0.09
Mukautuva exp-Golomb	x	ZOP	2.64	0.65
Mukautuva Rice		-	1.14	0.09
Mukautuva Rice		ZOP	2.29	0.55
Mukautuva Rice	x	-	1.14	0.10
Mukautuva Rice	x	ZOP	2.70	0.69

Taulukko 5.5: Mukautuvien universaalikoodien tulokset.

5.3 Häviöttömien menetelmien yhteenveto

Tulokset DPCM:lla olivat huomattavasti parempia kuin ilman DPCM:ia; tulokset kertovat tilastollisen mallintamisen tärkeydestä tiivistysjärjestelmässä. Entropiankoodaajat ovat järjestelmän "konehuone", mutta sen varsinainen älykkyys piilee tilastollisessa mallissa. Taulukossa 5.6 on esitetty menetelmät tiivistyssuhteen mukaan järjestyksessä.

Oletetusti aritmeettinen koodaus oli tehokkain ja Huffman-koodaus hyvin lähellä sitä. Staattinen aritmeettinen koodaus on tosin vain globaalisti optimaalinen; se ei hyödynnä paikallisia vaihteluita signaalin arvojen jakaumassa. Tästä syystä paikallisia vaihteluita hyödyntävät mukautuvat universaalikoodit pääsevät sen lähelle tuloksissa. Staattinen aritmeettinen koodaus ja Huffman-koodaus tarvitsevat koodauksessa kaksi signaalin läpikäyntiä ja mukautuvat universaalikoodit selviävät yhdellä läpikäynnillä.

Tulokset ovat lähellä Kosken EKG:tä koskevassa tutkimuksessa [23] vastaavilla menetelmillä 400 Hz:n taajuudella näytteistettyillä 13 bitin resoluution EKG-signaaleilla saavutettuja tiivistysuhteita (1.70-2.23). Tehokkaimmat tässä käytetyt menetelmät olivat tiivistyssuhteeltaan hieman parempia. Kyseiset menetelmät eivät olleet mukana Kosken tutkimuksessa.

Antoniolin ja Tonellan [2] tulokset 128 Hz:n taajuudella näytteistetystä 8 bitin

Menetelmä	Suodatettu	Ennustaja	μ_{CR}	σ_{CR}
Aritmeettinen koodaus		ZOP	2.29	0.53
Mukautuva Rice		ZOP	2.29	0.55
Huffman		ZOP	2.27	0.57
Mukautuva exp-Golomb		ZOP	2.23	0.53
exp-Golomb-koodi		ZOP	2.08	0.54
LZW		ZOP	1.87	0.57
LZSS		ZOP	1.02	0.13
Mukautuva Rice	x	ZOP	2.70	0.69
Aritmeettinen koodaus	x	ZOP	2.68	0.71
Huffman	x	ZOP	2.65	0.71
Mukautuva exp-Golomb	x	ZOP	2.64	0.65
exp-Golomb-koodi (k=2)	x	ZOP	2.45	0.54
γ-koodi	x	ZOP	2.31	0.82
LZW	x	ZOP	2.26	0.7
LZSS	x	ZOP	1.14	0.14

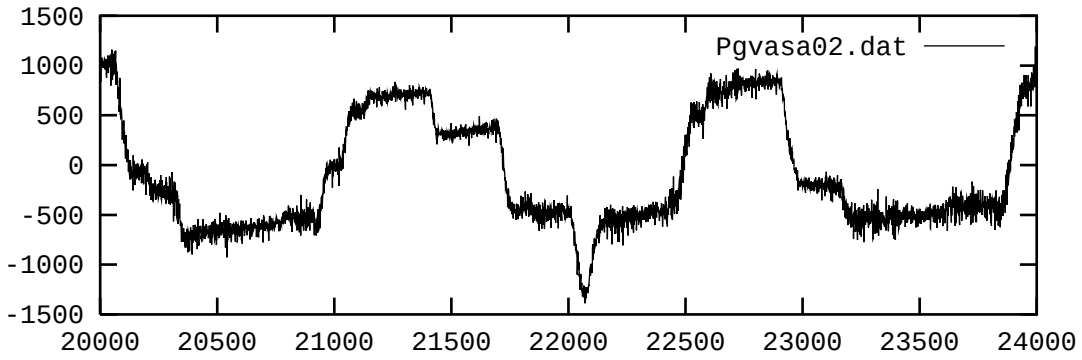
Taulukko 5.6: Häviöttömien menetelmien yhteenveto.

resoluution 20-kanavaisen EEG:n tiivistämisestä ovat 1.60-2.48, joista paras saatiin häviöttömällä DPCM:lla. Tulosten vertailukelpoisuus ei ole erityisen hyvä, koska näytteenottotaajuuksien ero on suuri ja voisi kuvitella, että 20-kanavainen signaali, joka mitataan samasta kohteesta sisältää varsin paljon redundanssia.

5.4 Häviölliset menetelmät

Häviöllisten menetelmien vertailuun ei riitä pelkkä tiivistyssuhde, vaan on sen lisäksi käytettävä jotain virhemetriikkaa. Menetelmiä vertaillaan siis tiivistyssuhteen ja laadun perusteella. Tässä käytettiin metriikoina MAE:ia, MSE:ia ja NRMSE:ia. Tuloksissa on esitetty häviöllisen menetelmän saavuttama keskimääräinen tiivistyssuhde μ_{CR} ja sitä vastaavat virhemetriikkojen keskiarvot μ_{MAE}, μ_{MSE} ja μ_{NRMSE}. Koska metriikat eivät täysin pysty karakterisoimaan rekonstruktion laatua, esitän muutamia kuvasarjoja rekonstruktion kehityksestä eri menetelmillä tiivistyssuhteen kasvaessa. Alkuperäinen signaalin lohko on esitetty kuvassa 5.1.

Kuten odottaa saattoi, skalaarikvantisointi ei päässyt laadullisesti kovin hyviin tuloksiin. On varsin todennäköistä, että 16-tasoisen kvantisoijan ulostulo on niin huonolaatuista, ettei sitä voi käyttää kliinisiin tarkoituksiin. Menetelmä on yksinkertainen ja se on mukana tutkimuksesssa lähinnä vertailun vuoksi. Skalaarikvantisoijan indeksi koodattiin sellaisenaan tiedostoon; parempia tiivistystuloksia saatai-



Kuva 5.1: Alkuperäinen signaali.

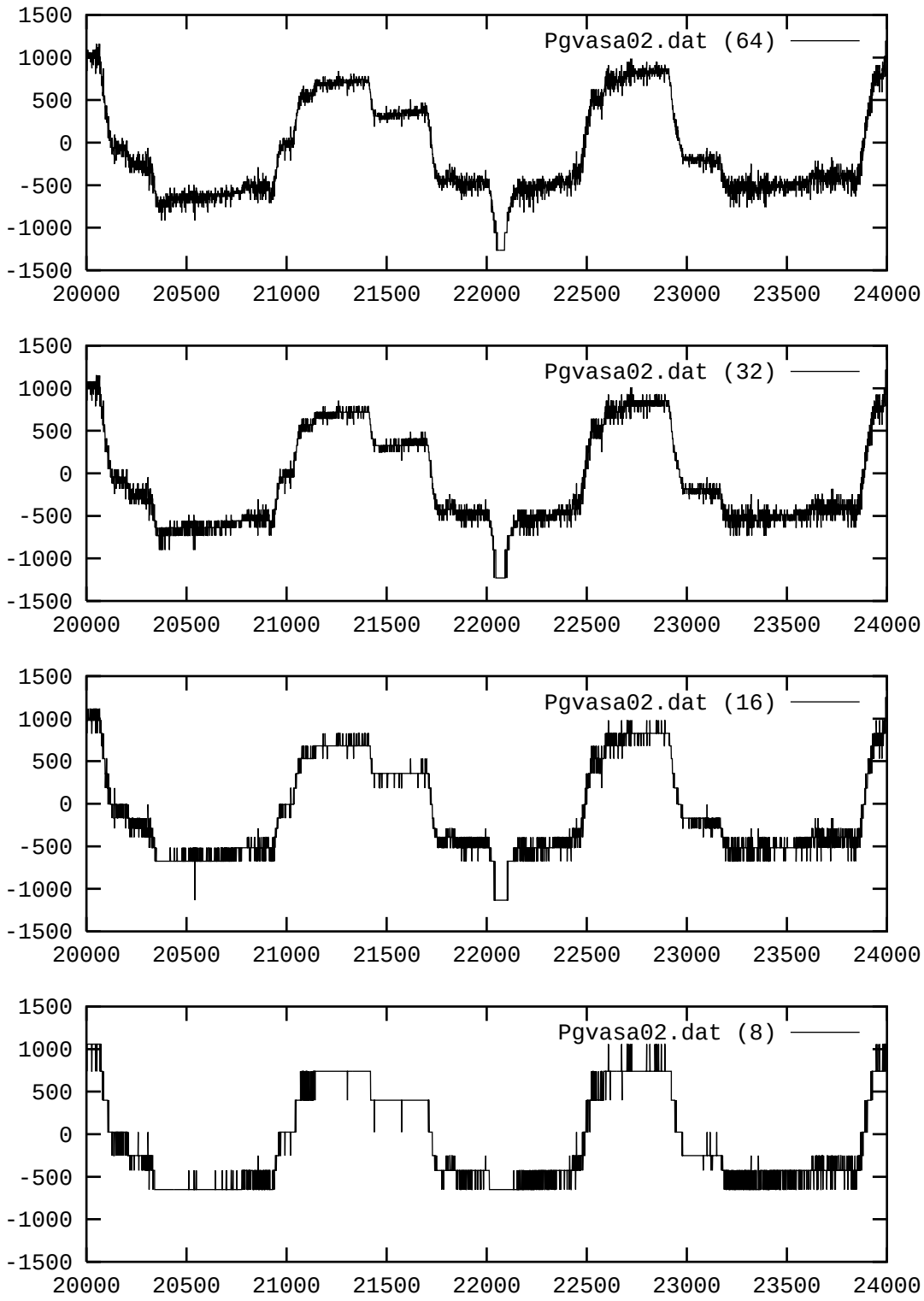
siin luultavasti koodaamalla indeksit esimerkiksi LZW-algoritmilla, koska indeksien välillä on edelleen riippuvuuksia. Kuvassa 5.3 on esitetty skalaarikvantisoinnin rekonstruktioita. Kvantisointi suoritettiin 64:llä (CR = 2.2, NRMSE = 0.2), 32:lla (CR = 2.6, NRMSE = 0.03), 16:lla (CR = 3.2, NRMSE = 0.06) ja 8:lla (CR = 4.3, NRMSE = 0.11) tasolla. Tulokset on esitetty taulukossa 5.7.

Tasot	μ_{CR}	μ_{MAE}	μ_{MSE}	μ_{NRMSE}
512	1.40	0.9	3.3	0.002
256	1.60	1.9	9.8	0.003
128	1.84	3.9	32.0	0.006
64	2.15	7.8	113.2	0.013
32	2.58	15.3	429.0	0.026
16	3.23	29.6	1623.0	0.050
8	4.3	59.2	6594.0	0.100
4	6.47	132.0	34700.0	0.220

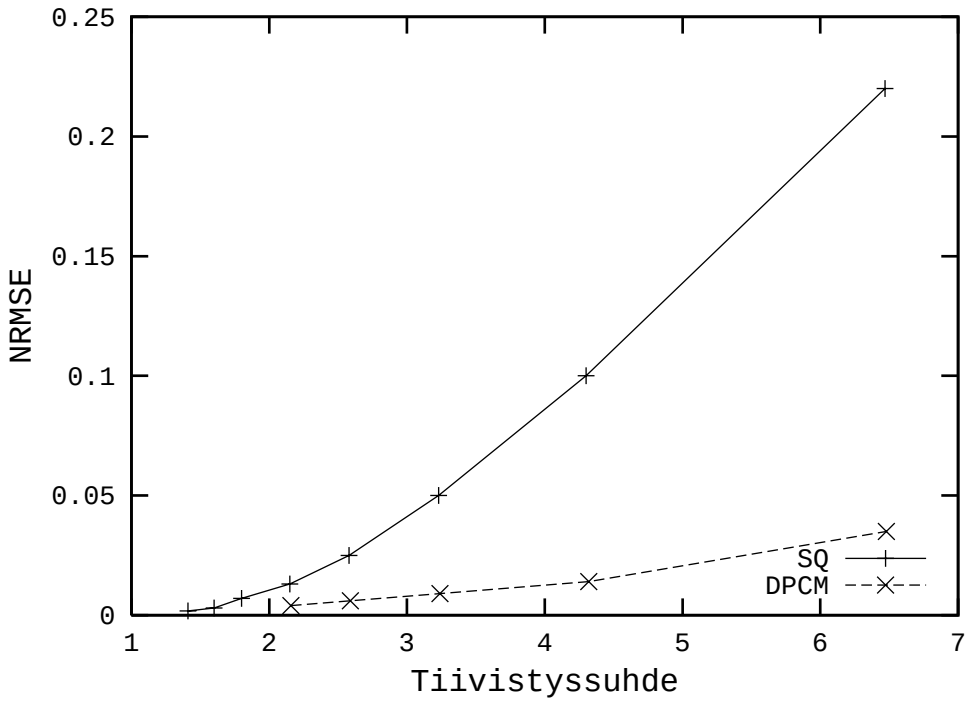
Taulukko 5.7: Skalaarikvantisoinnin tulokset.

Häviöllisen DPCM:n ennustajana käytettiin ZOP:ia ja kvantisoija suunniteltiin Lloydin algoritmilla ZOP:n alkuperäisen signaalin ennustusvirhesignaalille. Menetelmä on pienillä tiivistyssuhteilla täysin vertailukelpoinen monimutkaisemman muunnoskoodauksen kanssa. DPCM:n laatu kuitenkin heikkenee nopeasti, kun tasojen määrää pienennetään; neljätasoinen DPCM on jo selkeästi laadultaan huonompi kuin kynnystetyt muunnokset, joilla saavutetaan sama tiivistyssuhde. Häviöllisen DPCM:n tulokset on esitetty taulukossa 5.8. Skalaarikvantisoinnin ja DPCM:n tuloksista on yhteenveto kuvassa 5.3.

Kuvassa 5.4 on esitetty DPCM:n rekonstruktioita. Tiivistyssuhteet ja NRMSE:t



Kuva 5.2: Skalaarikvantisoinnin rekonstruktioita.



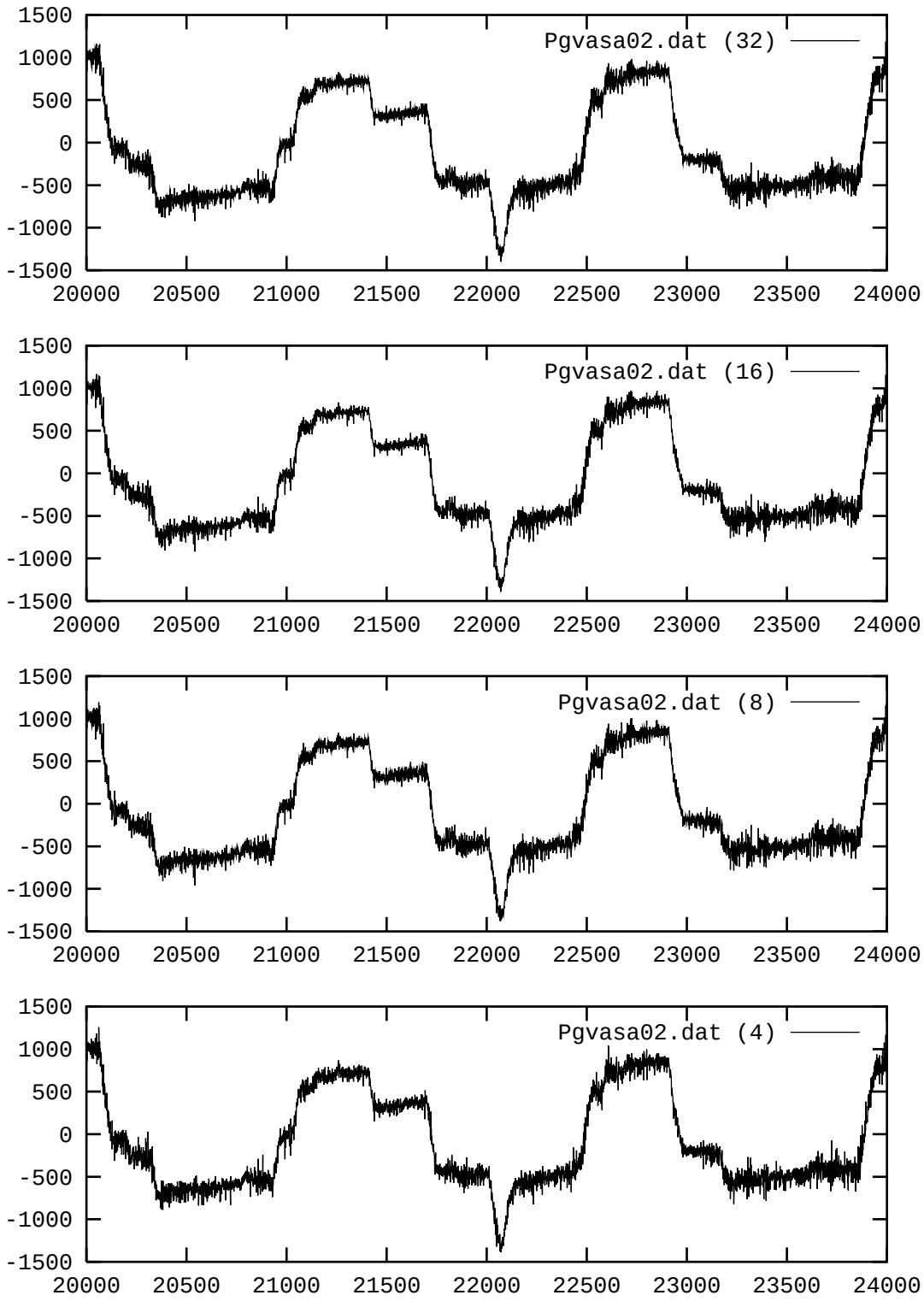
Kuva 5.3: Skalaarikvantisoinnin ja DPCM:n tulokset.

ylhäältä alas ovat CR = 2.7 ja NRMSE = 0.01, CR = 3.2 ja NRMSE = 0.02, CR = 4.5 ja NRMSE = 0.04 ja CR = 6.8 ja NRMSE = 0.07. Rekonstruktiot ovat varsin laadukkaita. Kuvista näkee sen, että DPCM lisää kohinaa jonkin verran. Menetelmää ei voida käyttää suuremman tiivistyssuhteen saavuttamiseksi muuten kuin yhdistämällä se johonkin entropiankoodaajaan. Vähemmillä kvantisointitasoilla virhe tulee huomattavan suureksi.

Tasot	μ_{CR}	μ_{MAE}	μ_{MSE}	μ_{NRMSE}
64	2.16	0.9	63.7	0.004
32	2.59	1.7	89.1	0.007
16	3.24	3.0	127.9	0.009
8	4.32	5.1	217.6	0.014
4	6.48	11.1	1068.0	0.035

Taulukko 5.8: Häviöllisen DPCM:n tulokset.

Testasin muunnosten keskinäistä paremmuutta kvantiilikynnystämisen avulla. Lisäsin jokaisen muunnoksen samaan ohjelmarunkoon, jossa ensin jaettiin syöte 512 näytteen lohkoihin. Kullekin lohkolle tehtiin muunnos ja siitä kynnystettiin pois tiet-



Kuva 5.4: DPCM:n rekonstruktioita.

ty määrä kertoimia. Kynnystetty data koodattiin lopuksi tiedostoon aritmeettisella koodauksella. Muunnosten väliset tulokset ovat näin vertailukelpoisia. Aaltopakettimuunnos tarvitsee lisäksi tiedon siitä, mikä mahdollisista kannoista kullekin lohkolle on valittu; tämä on myös mukana tiivistystuloksissa. Aalloke- ja aaltopakettimuunnoksissa käytin aallokkeena Daubechies-4-aalloketta [12]. Muunnosten tehoon voidaan jonkin verran vaikuttaa valitsemalla sopiva kanta; tekemieni testien perusteella erot kantojen[1] välillä ovat pieniä. Kvantiilikynnystettyjen muunnosten tulokset on esitetty taulukoissa 5.9-5.13 ja yhteenveto niistä on esitetty kuvassa 5.5.

Kuvasta 5.5 huomataan, että pienillä tiivistyssuhteilla Hartley-muunnosta lukuunottamatta muunnokset ovat melko tasaväkisiä. Walsh-Hadamard-muunnos on muita muunnoksia hieman parempi pienillä tiivistyssuhteilla; syy on luultavasti se, että muunnoksen ulostulo sopii paremmin aritmeettisen koodaajan koodattavaksi kuin muiden muunnosten. Pienillä tiivistyssuhteilla aaltopakettimuunnos on hieman parempi kuin aallokemuunnos; tämä oli odotettavaa, sillä onhan se teknisesti kehittyneempi. Suuremmilla tiivistyssuhteilla aallokemuunnos on parempi. Tämä luultavasti johtuu siitä, että aaltopakettimuunnos tarvitsee sivuinformaatiota puun rakenteesta ja signaalin informaatio on luultavasti keskittynyt matalille taajuuksille, jolloin aallokemuunnos tehoaa siihen hyvin.

Kuvissa 5.6 ja 5.7 on esitetty kvantiilikynnystyksen ja muunnoskoodauksen rekonstruktioita aalloke- ja kosinimuunnoksella. Aallokemuunnoksen tiivistyssuhteet ja NRMSE:t ylhäältä alas ovat CR = 2.2 ja NRMSE = 0.03, CR = 7.2 ja NRMSE = 0.09, CR = 12.7 ja NRMSE = 0.10 ja CR = 23.1 ja NRMSE = 0.15. Vastaavasti kosinimuunnoksella arvot olivat CR = 2.3 ja NRMSE = 0.03, CR = 7.6 ja NRMSE = 0.09, CR = 12.7 ja NRMSE = 0.11 ja CR = 23.1 ja NRMSE = 0.12. Poistettujen kertoimien osuus on ilmoitettu kuvissa. Kuvan 5.6 alimman signaalin piikit ovat reunaepäjatkuvuuksia.

Peräkkäisten approksimaatioiden kvantisointi toimii huomattavan paljon paremmin kuin kvantiilikynnystäminen. Tulokset on esitetty taulukoissa 5.14-5.17 ja yhteenveto kuvassa 5.8; BPS on käytettyjen bittien määrä näytettä kohti (bits per sample). Kosini-, aalloke-, aaltopakettimuunnokset hyötyvät SAQ:n käytöstä, mutta Walsh-Hadamard-muunnoksen tiivistystulos huononee. Tämä luultavasti johtuu siitä, että Walsh-Hadamard-muunnoksen tuottamat arvot sopivat paremmin aritmeettiselle koodaukselle. Toinen mahdollinen syy on se, että koska Walsh-Hadamard-muunnoksen tuottamat komponentit eivät ole muunnoksen kantafunktioiden nollan-

[1] Testasin eri Daubechies-, Coiflet- ja Symlet-aallokkeita.

Kynnystys	μ_{CR}	μ_{MAE}	μ_{MSE}	μ_{NRMSE}
60%	2.7	8.2	395.7	0.021
70%	3.3	10.3	682.5	0.028
80%	4.4	13.3	1212.0	0.038
90%	7.3	19.3	2564.0	0.055
95%	12.4	28.4	4943.0	0.077
97%	18.0	38.0	7813.0	0.098
98%	24.0	28.0	11200.0	0.118
99%	38.5	74.0	22000.0	0.166

Taulukko 5.9: Hartley-muunnoksen ja kynnystämisen tulokset.

Kynnystys	μ_{CR}	μ_{MAE}	μ_{MSE}	μ_{NRMSE}
60%	3.1	4.7	89.4	0.009
80%	4.8	8.5	282.7	0.017
90%	7.9	12.1	570.6	0.023
95%	13.0	16.8	1064.0	0.033
97%	18.5	22.5	1762.0	0.045
98%	24.5	29.5	2931.0	0.059
99%	38.8	49.5	8503.0	0.098

Taulukko 5.10: Diskreetin kosinimuunnoksen ja kynnystämisen tulokset.

Kynnystys	μ_{CR}	μ_{MAE}	μ_{MSE}	μ_{NRMSE}
60%	8.1	8.32	157.6	0.018
90%	10.9	16.2	913.2	0.031
95%	16.8	25.5	2411.0	0.05
97%	24.0	35.5	5228.0	0.07
98%	31.9	50.5	9809.0	0.10
99%	51.0	80.5	25480.0	0.16

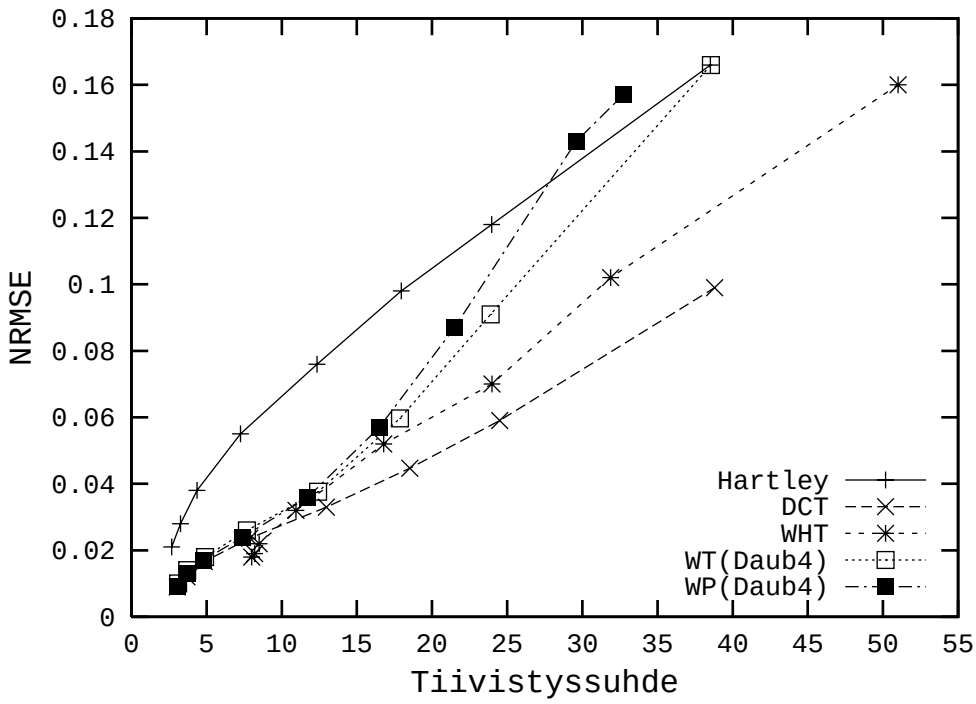
Taulukko 5.11: Walsh-Hadamard -muunnoksen ja kynnystämisen tulokset.

Kynnystys	μ_{CR}	μ_{MAE}	μ_{MSE}	μ_{NRMSE}
70%	3.8	7.36	185.5	0.014
80%	4.9	9.8	326.2	0.018
90%	7.7	13.74	651.0	0.026
95%	12.4	19.3	1288.0	0.038
97%	17.9	29.5	3012.0	0.059
98%	23.9	42.0	7029.0	0.091
99%	38.6	78.3	24530.0	0.166

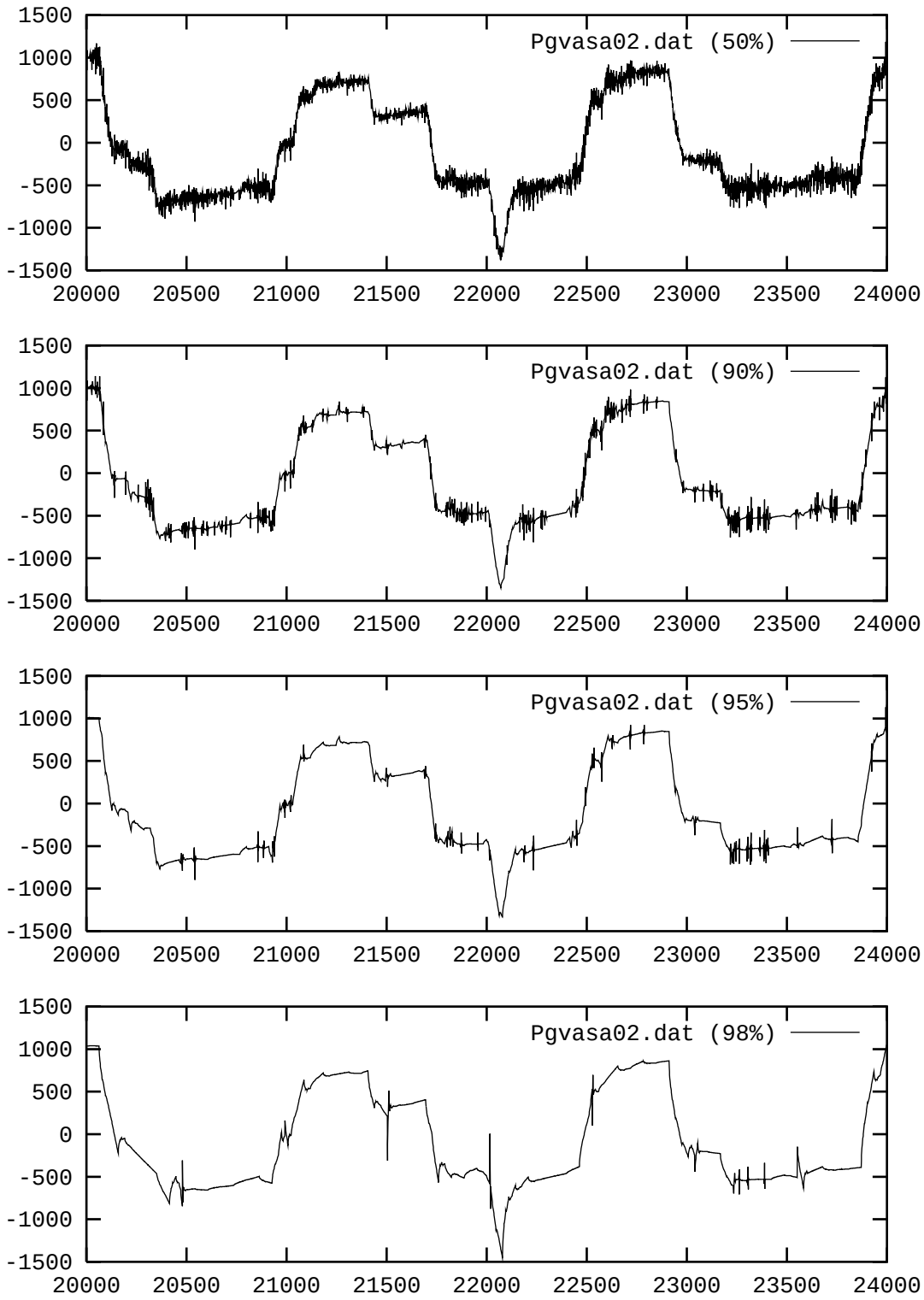
Taulukko 5.12: Aallokemuunnoksen ja kynnystämisen tulokset.

Kynnystys	μ_{CR}	μ_{MAE}	μ_{MSE}	μ_{NRMSE}
60%	3.1	4.9	80.8	0.009
70%	3.7	6.6	146.4	0.013
80%	4.8	9.0	267.3	0.017
90%	7.4	7.4	557.5	0.024
95%	11.8	18.5	1151.0	0.036
97%	16.5	28.0	2725.0	0.057
98%	21.5	40.2	6327.0	0.087
99%	32.7	74.3	21570.0	0.157

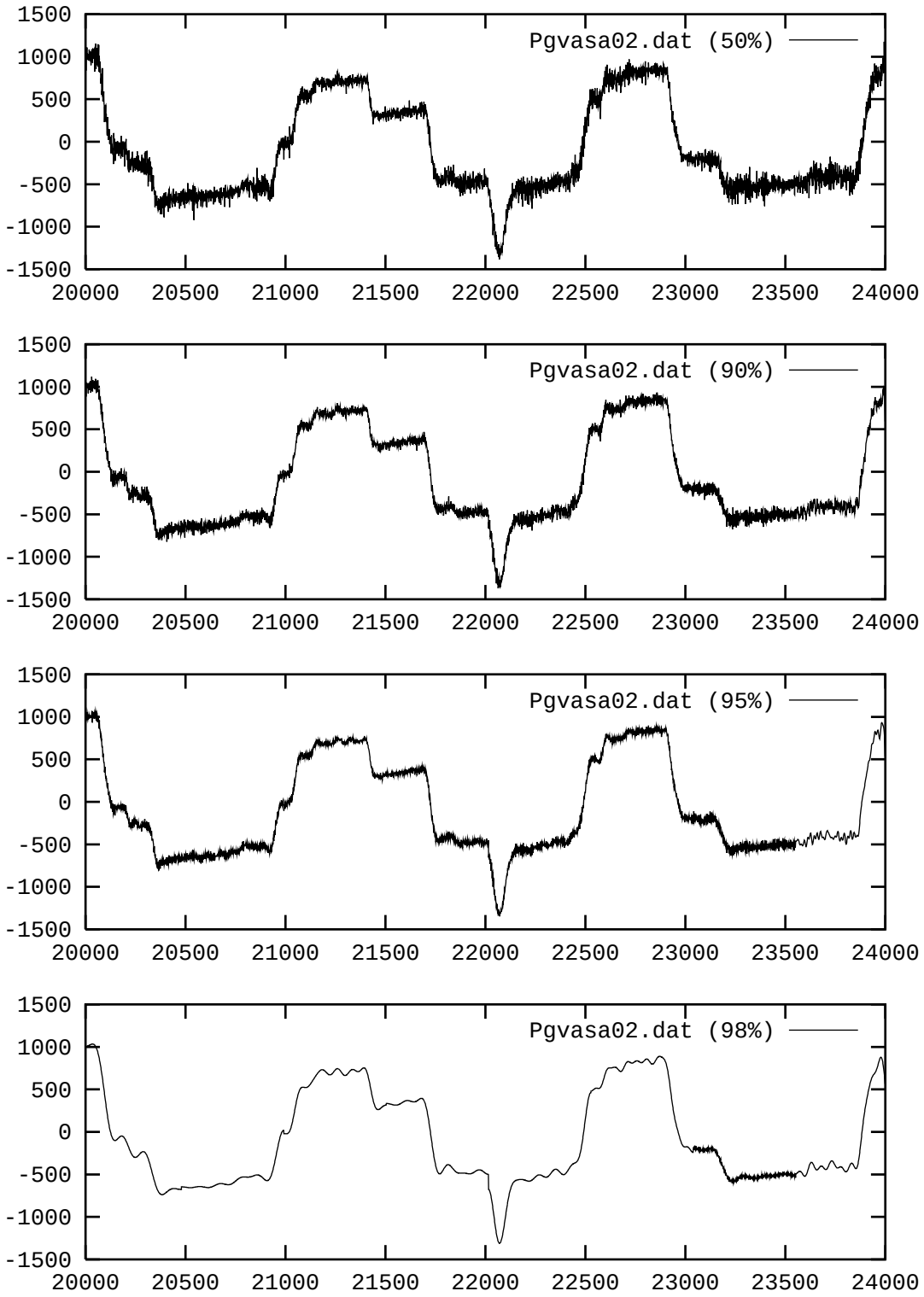
Taulukko 5.13: Aaltopakettimuunnoksen ja kynnystämisen tulokset.



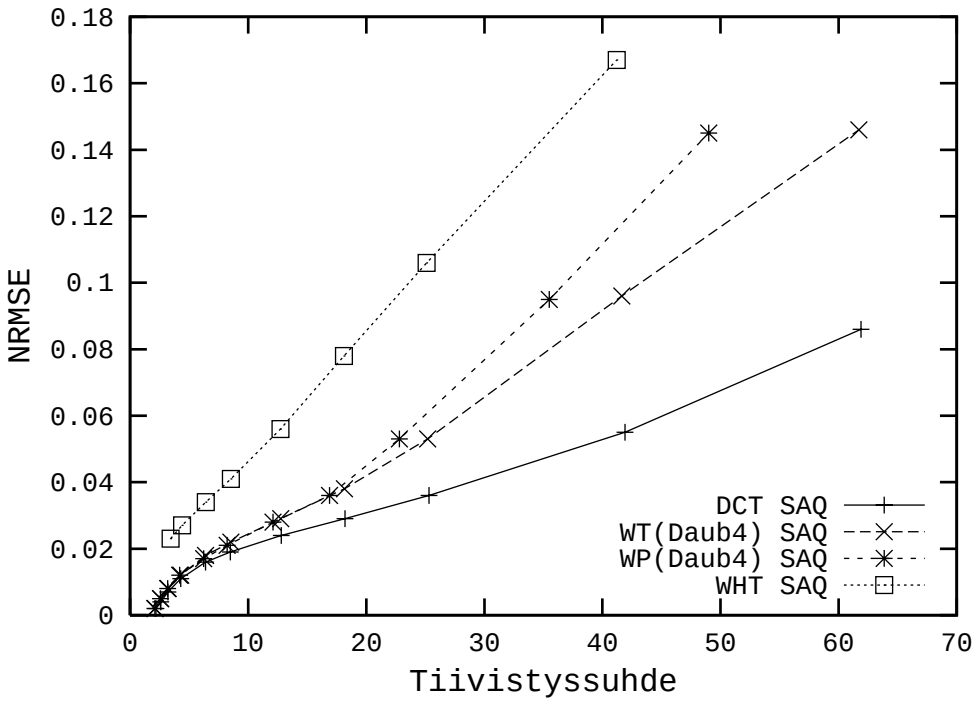
Kuva 5.5: Kvantiilikynnystyksen tulokset.



Kuva 5.6: Aallokemuunnoksen(Daub4) ja kvantiilikynnystyksen rekonstruktioita.



Kuva 5.7: Diskreetin kosinimuunnoksen ja kvantiilikynnystyksen rekonstruktioita.



Kuva 5.8: Peräkkäisten approksimaatioiden kvantisoinnin variantin tulokset.

ylitysten lukumäärän mukaan järjestyksessä eli samankaltaiset kantafunktiot eivät ole lähekkäin, jolloin merkittävät kertoimet klusteroidu. Tässä tapauksessa peräkkäisten approksimaatioiden kvantisoinnin variantti ei toimi niin hyvin. DCT osoittautuu parhaaksi muunnokseksi näillä tiivistysmenetelmillä ja -metriikoilla.

Kuvissa 5.9 ja 5.10 on kuvattu peräkkäisten approksimaatioiden kvantisoinnin ja aalloke- ja kosinimuunnoksen rekonstruktioita. Aallokemuunnoksen tiivistysuhteet ja NRMSE:t ylhäältä alas ovat CR = 4.3 ja NRMSE = 0.04, CR = 8.6 ja NRMSE = 0.08, CR = 18.3 ja NRMSE = 0.11 ja CR = 25.4 ja NRMSE = 0.12. Vastaavat arvot kosinimuunnokselle ovat CR = 4.3 ja NRMSE = 0.04, CR = 8.6 ja NRMSE = 0.08, CR = 18.3 ja NRMSE = 0.11 ja CR = 25.5 ja NRMSE = 0.11. Rekonstruktioiden ulkonäössä on selvä ero, vaikka niiden virheet ovat lähes samat.

5.5 Häviöllisten menetelmien yhteenveto

Muunnoskoodaus osoittautui testatuista menetelmistä parhaaksi ja muunnoksista parhaaksi diskreetti kosinimuunnos. Muunnosten tiivistämisen menetelmistä peräkkäisten approksimaatioiden kvantisointi on selvästi parempi kuin kvantiilikynnystäminen. Matalilla tiivistyssuhteilla DPCM saavuttaa varsin laadukkaan tuloksen, joka

BPS	μ_{CR}	μ_{MAE}	μ_{MSE}	μ_{NRMSE}
6.5	3.4	6.0	179.0	0.023
4.0	3.7	8.4	256.9	0.025
3.0	4.4	11.6	412.7	0.028
2.0	6.4	16.3	814.2	0.034
1.5	8.5	20.3	1333.0	0.41
1.0	12.8	28.3	2729.0	0.056
0.7	18.1	38.9	5473.0	0.078
0.5	25.0	53.7	10530.0	0.106
0.3	41.6	87.5	26860.0	0.167

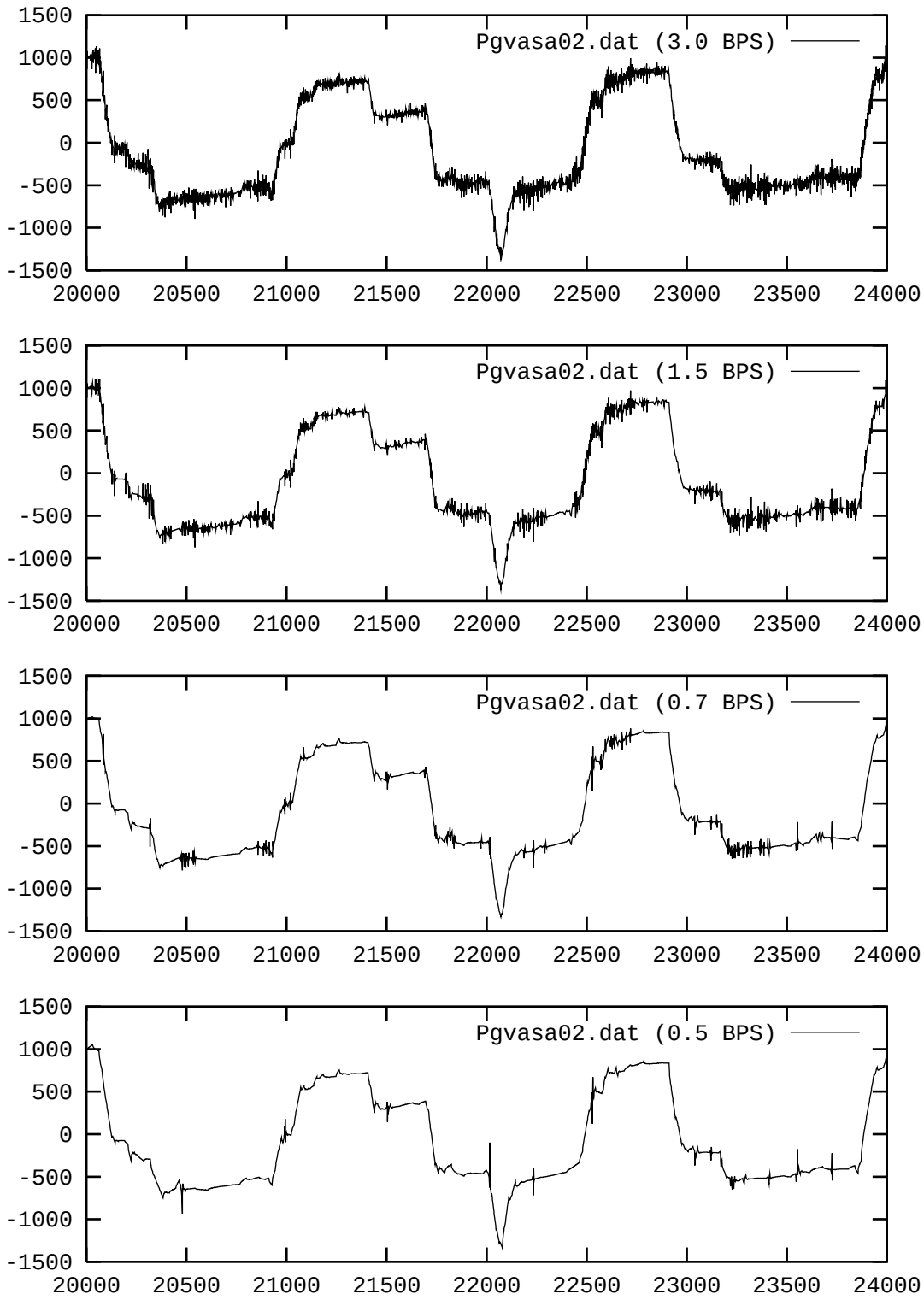
Taulukko 5.14: Walsh-Hadamard-muunnoksen ja SAQ:n tulokset.

BPS	μ_{CR}	μ_{MAE}	μ_{MSE}	μ_{NRMSE}
6.5	2.1	1.06	3.2	0.002
4.0	3.2	3.6	44.1	0.007
3.0	4.3	5.4	108.2	0.011
2.0	6.4	8.0	240.9	0.016
1.5	8.5	9.7	364.3	0.019
1.0	12.2	12.2	570.0	0.024
0.7	18.2	14.7	815.7	0.029
0.5	25.3	18.0	1187.0	0.036
0.3	41.9	27.5	2508.0	0.055

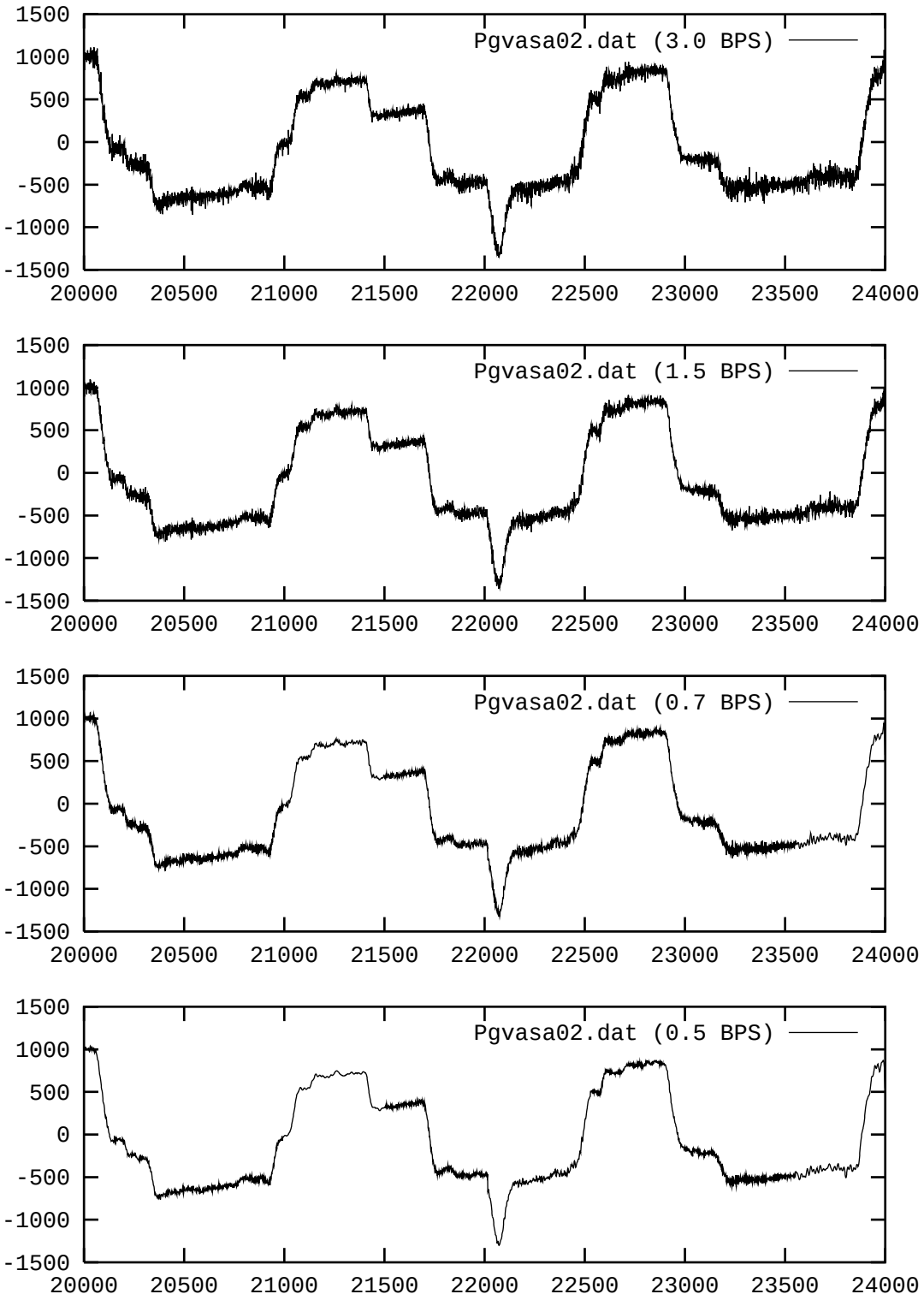
Taulukko 5.15: Diskreetin kosinimuunnoksen ja SAQ:n tulokset.

BPS	μ_{CR}	μ_{MAE}	μ_{MSE}	μ_{NRMSE}
6.5	2.1	1.2	4.0	0.002
5.0	2.6	2.6	19.8	0.005
4.0	3.2	4.2	54.4	0.008
3.0	4.3	6.6	135.3	0.012
2.0	6.4	9.8	302.6	0.018
1.5	8.5	11.9	455.9	0.022
1.0	12.8	15.3	758.3	0.029
0.7	18.2	19.6	1194.0	0.038
0.5	25.2	26.8	2256.0	0.054
0.3	41.6	45.9	7412.0	0.096

Taulukko 5.16: Aallokemuunnoksen ja SAQ:n tulokset.



Kuva 5.9: Aallokemuunnoksen(Daub4) ja SAQ:n rekonstruktioita.



Kuva 5.10: Diskreetin kosinimuunnoksen ja SAQ:n rekonstruktioita.

BPS	μ_{CR}	μ_{MAE}	μ_{MSE}	μ_{NRMSE}
6.5	2.1	1.2	3.8	0.002
5.0	2.6	2.5	18.7	0.005
4.0	3.2	4.0	50.2	0.008
3.0	4.2	6.3	121.7	0.012
2.0	6.2	9.3	273.7	0.017
1.5	8.2	11.4	414.6	0.021
1.0	12.1	14.6	688.3	0.028
0.7	16.9	19.0	1122.0	0.036
0.5	22.8	26.4	2186.0	0.053
0.3	35.5	45.4	7271.0	0.095

Taulukko 5.17: Aaltopakettimuunnoksen ja SAQ:n tulokset.

on vertailukelpoinen muunnoskoodien kanssa.

Jaladeddine *et al.* [18] esittävät yhteenvedon aiemmasta tiivistystutkimuksesta EKG:n tapauksessa. Tulosten vertailu on vaikeaa, koska joissakin tutkimuksissa ei tuloksissa esitetty virhettä sopivalla virhemetriikkalla. Verrattuna niihin menetelmiin, joista virhe oli esitetty, saavuttaa tämän tutkimuksen muunnoskoodaus huomattavasti parempia tuloksia. On muistettava, että kyse on erilaisista signaaleista ja eri näytteenottotaajuuksista ja joissakin tapauksissa signaaleilla oli lisäksi monta kanavaa.

Hilton [15] esittää tulokset aalloke- ja aaltopakettimuunnoksen käytöstä EKG-signaalien tiivistämiseen. Tutkimuksessa käytettiin muunnoksen koodaamiseen EZW-koodaajaa (embedded zerotree wavelet), joka on suunniteltu ottamaan huomioon aallokemuunnoksen hierarkkinen rakenne. EZW-koodaaja on eräänlainen peräkkäisten approksimaatioiden kvantisoija. Tulokset aalloke- ja aaltopakettimuunnosten välillä ovat samanlaisia, kuin tässäkin tutkimuksessa. Vaikka aaltopakettimuunnos on kehittyneempi, sen tiivistysuhde luultavasti kärsii sivuinformaatiosta sitä enemmän, mitä suuremmaksi tiivistyssuhde kasvaa. Tämän tutkimuksen tulokset peräkkäisten approksimaatioiden kvantisoinnin variantilla ovat hieman parempia. Osaltaan asiaan vaikuttaa se, että Hiltonin tutkimuksessa tulokset esitettiin MIT-BIH EKG-testitietokannalle, jota pidetään yleisesti vaikeana tiivistää.

6 Yhteenveto

Tutkimuksessa saavutettiin muiden biosignaaleiden tiivistyksen tuloksiin verrattuna hyviä tuloksia ja selvitettiin, mitkä menetelmät sopivat silmänliikesignaaleiden tiivistämisessä hyvin. Tutkimuksen aikana kehitettiin kaksi uutta tiivistysmenetelmää, yksi häviötön ja yksi häviöllinen, jotka ovat yhtä hyviä kuin parhaat tutkimuksessa käsitetellyt tunnetut menetelmät. Joitakin tunnettuja menetelmiä jäi ajanpuutteen vuoksi toteuttamatta.

Häviöllisen tiivistyksen tapauksessa ei tehty sovelluskohtaista arviointia laadusta, mutta analyysi on tarkoitus tehdä tulevaisuudessa. Tässä kehitettyjen parhaiden menetelmien laatu on luultavasti objektiivisten metriikoiden ja visuaalisen tarkastelun perusteella hyvä. Laadulla tässä tarkoitetaan sitä, että tiivistys ei merkittävästi muuta signaalista laskettavia lääketieteellisesti tärkeitä arvoja. Jotkut arvot ovat herkkiä pienillekin muutoksille signaalissa, jolloin virherajat on syytä asettaa tiukoiksi.

Yksi mahdollinen jatkotutkimuksen aihe on kehittää menetelmiä, jotka pystyvät mukautumaan virherajoihin. Joitakin signaaleita voidaan tiivistää enemmän kuin toisia ylittämättä kuitenkaan virherajoja. Tätä voitaisiin hyödyntää keskimäärin suuremman tiivistystehon saavuttamiseksi laatua menettämättä. Sopivien virherajojen etsiminen on tehtävä empiiriisesti tai asetettava niin tiukaksi, että objektiivinen virhemetriikka pakottaa laadun hyväksi.

Jatkossa tutkimuksen aikana kehitettyille ja toteutetuille menetelmille on tarkoitus tehdä yksityiskohtainen virheanalyysi kuuloherätevastesignaaleiden (auditory brainstem response, ABR) tapauksessa, jossa tutkitaan nimenomaan sitä, kuinka hyvin objektiivisia virhemetriikoita voidaan soveltaa laadunarvioinnissa. Toinen jatkotutkimuksen aihe on paljon tilaa vievien positroniemissiotomografiakuvien (PET) tiivistäminen.

Viitteet

[1] S. T. Alexander. *Adaptive Signal Processing.* Springer-Verlag, New York, 1986.

[2] G. Antoniol and P. Tonella. EEG data compression techniques. *IEEE Transactions on Biomedical Engineering*, 44(2):105–114, February 1997.

[3] E. O. Brigham. *The Fast Fourier Transform and Its Applications.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1988.

[4] M. Burrows and D. J. Wheeler. A block sorting lossless data compression algorithm. Research report, Digital Systems Research Center, May 1994.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, Massachusetts, 1990.

[6] D. J. Defatta, J. G. Lucas, and W. S. Hodgkiss. *Digital Signal Processing: A System Design Approach.* John Wiley & Sons, 1998.

[7] P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 11:194–203, 1975.

[8] M. Erne. Digital audio compression algorithms. In *DAFX98 Proceedings*, 1998. Saatavilla WWW:stä, http://www.iua.upf.es/dafx98/papers/ERN17.ps, viitattu 25.9.1999.

[9] M. Erne. Embedded audio compression using wavelets and improved psychoacoustic models. In *DAFX98 Proceedings*, 1998. Saatavilla WWW:stä, http://www.iua.upf.es/dafx98/papers/ERN16.ps, viitattu 25.9.1999.

[10] P. M. Fenwick. A new data structure for cumulative frequency tables. *Software - Practice and Experience*, 24(3):327–336, March 1994.

[11] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, Boston, 1992.

[12] A. L. Graps. An introduction to wavelets. *IEEE Computational Sciences and Engineering*, 2(2):50–61, 1995. Saatavilla WWW:stä, http://www.amara.com/IEEEwave/IEEEwavelet.html, viitattu 22.9.1999.

[13] S. S. Haykin. *Introduction to Adaptive Filters*. Macmillan Publishing Company, New York, 1984.

[14] R. Hecht-Nielsen. *Neurocomputing*. Addison-Wesley, Reading, Massachusetts, 1990.

[15] M. L. Hilton. Wavelet and wavelet packet compression of electrocardiograms. *IEEE Transactions on Biomedical Engineering*, 44(5):394–402, May 1997.

[16] P. Howard and J.S. Vitter. Practical implementations of arithmetic coding. In J. A. Storer, editor, *Image and Text Compression*, pages 85–112. Kluwer Academic Publishers, Norwell, Massachusetts, 1992.

[17] P. G. Howard and J. S. Vitter. Fast and efficient lossless image compression. In J. A. Storer and M. Cohn, editors, *Proc. Data Compression Conference*, pages 351–360, Utah, April 1993. Snowbird. Saatavilla WWW:stä, http://www.cs.duke.edu/~jsv/Papers/catalog/node55.html, viitattu 20.9.1999.

[18] S.M.S. Jaladeddine, C.G. Hutchens, R.D. Strattan, and W.A. Coberly. ECG data compression techniques - a unified approach. *IEEE Transactions on Biomedical Engineering*, 37(4):329–343, April 1990.

[19] M. Juhola. *On Computer Analysis of Digital Signals Applied to Eye Movements*. PhD thesis, University of Turku, 1987.

[20] M. Juhola. Effects of digital lowpass filtering on the parameters of nystagmus eye movements. *Medical Progress through Technology*, 17:111–118, 1991.

[21] M. Juhola. Median filtering is appropriate to signals of saccadic eye movements. *Computers in Biology and Medicine*, 21(1):43–49, 1991.

[22] M. Juhola, I. Pyykkö, H. Aalto, and T. Hirvonen. Effects of digital filtering on the parameters of vestibulo-ocular reflex signals. *Automedica*, 16:205–216, 1998.

[23] A. Koski. *On Structural Recognition and Analysis Methods Applied to ECG Signals*. PhD thesis, University of Turku, 1997.

[24] P. Kraniauskas. *Transforms in Signals and Systems.* Addison-Wesley, Reading, Massachusetts, 1992.

[25] T. J. Lynch. *Data Compression Techniques and Applications.* Lifetime Learning Publications, Belmont, California, 1985.

[26] R. G. Lyons. *Understanding Digital Signal Processing.* Addison-Wesley, Reading, Massachusetts, 1996.

[27] A. Moffat. Arithmetic coding revisited. *ACM Transactions on Information Systems*, 16(3):256–294, July 1998.

[28] S. J. Orfanidis. *Optimum Signal Processing: An Introduction.* Macmillan Publishing Company, New York, 1985.

[29] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: the Art of Scientific Computing.* Cambridge University Press, 1992. Saatavilla WWW:stä, http://cfatab.harvard.edu/nr/nronline.html, viitattu 22.9.1999.

[30] J. G. Proakis and D. G. Manolakis. *Digital Signal Processing Principles, Algorithms, and Applications.* Prentice-Hall, Upper Saddle River, New Jersey, 1996.

[31] J. Rissanen and G.G. Langdon. Arithmetic coding. *IBM Journal of Research and Development*, 23(2):149–162, March 1979.

[32] A. Said and W. A. Pearlman. A new fast and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6:243–250, June 1996. Saatavilla WWW:stä, ftp://ipl.rpi.edu/pub/EW_Code/SPIHT.ps.gz, viitattu 5.7.1999.

[33] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication.* The University of Illinois Press, Urbana, 1963.

[34] J. A. Storer. *Data Compression: Methods and Theory.* Computer Science Press, Rockville, Maryland, 1988.

[35] J.A. Storer and T.G. Szymanski. Text compression via textual substitution. *Journal of the ACM*, 29(4):928–951, 1982.

[36] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, Massachusetts, 1997.

[37] F. J. Taylor. *Digital Filter Design Handbook*. Marcel Dekker, New York, 1983.

[38] J. Teuhola. A compression method for clustered bit-vectors. *Information Processing Letters*, 7(6):308–311, October 1978.

[39] T. Tossavainen, M. Juhola, and A. Koski. Lossless compression of eye movement signals. In P. Kokol, B. Zupan, J. Stare, M. Premik, and R. Engelbrecht, editors, *Medical Informatics Europe '99*, pages 369–373. IOS Press, 1999.

[40] E. W. Weisstein. *CRC Concise Encyclopaedia of Mathematics*. CRC Press, November 1998. Saatavilla WWW:stä, http://www.treasure-troves.com/math/, viitattu 20.9.1999.

[41] T. Welch. A technique for high performance data compression. *IEEE Computer*, 17(6):8–19, June 1984.

[42] M. V. Wickerhauser. Lectures on wavelet packet algorithms. Technical report, Washington University, November 1991. Saatavilla WWW:stä, http://wuarchive.wustl.edu/doc/techreports/wustl.edu/math/papers/inria300.ps.Z, viitattu 5.7.1999.

[43] I.H. Witten, R.M. Neal, and J.G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, June 1987.

[44] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-12(3):337–343, May 1977.

[45] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, IT-24(5):530–536, September 1978.